

华为认证 AI 系列教程

HCIP-AI-Model Developer

编译优化

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

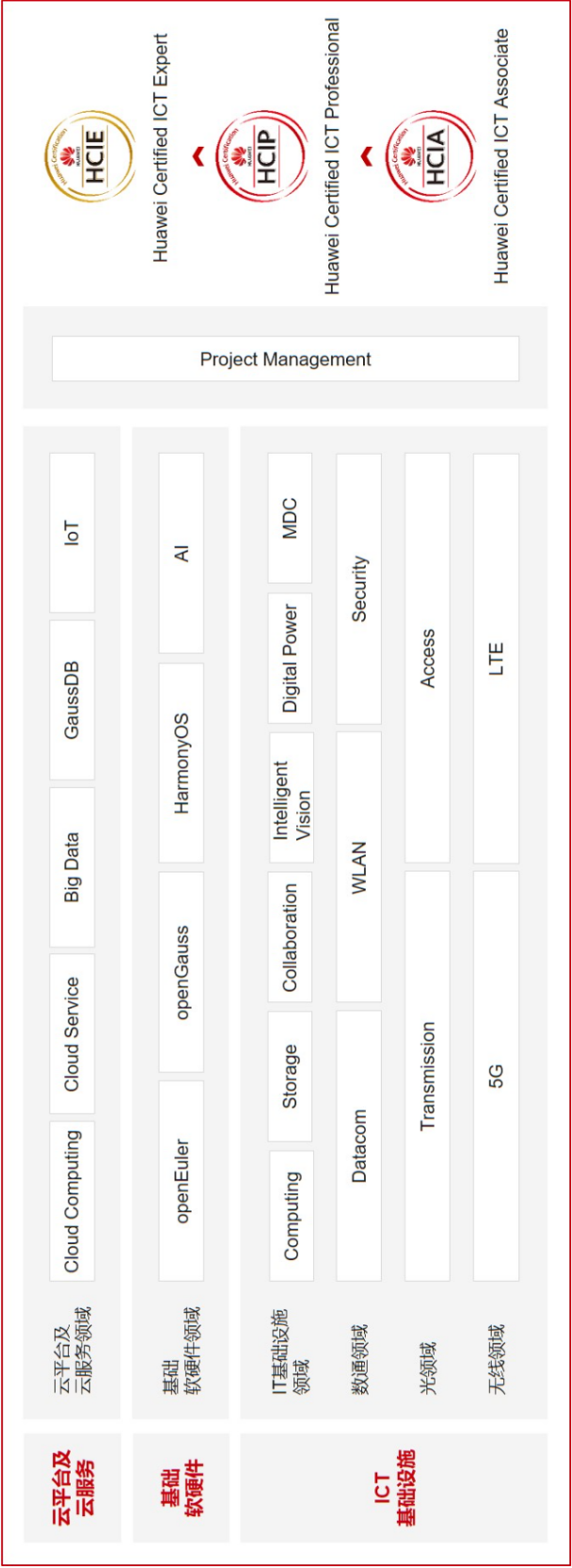
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

本实验指导书共包含 2 个小的实验。

- 实验一 torch.jit，通过 PyTorch 中的 jit 模块实现加速优化。
- 实验二 torch.compile，通过 PyTorch 中的 compile 模块实现加速优化。

读者知识背景

本课程为华为认证课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有 Python 编程基础知识；
- 具备线性代数、概率论等基础数学知识。

实验环境说明

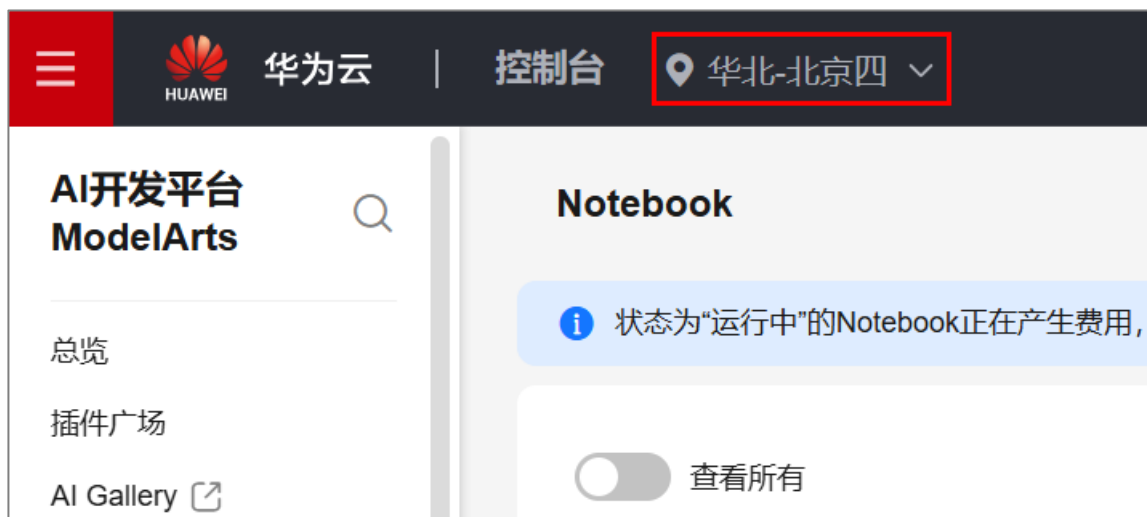
本实验环境为 PyTorch2.1。

步骤 1 点击以下链接进入 ModelArts 服务，然后点击控制台进入到后台管理界面

<https://www.huaweicloud.com/product/modelarts.html>



步骤 2 点击左上角切换到北京四区域



步骤 3 添加授权

如果是第一次使用 ModelArts 服务，需要进行授权。



点击创建后完成授权。

步骤 4 点击左侧开发空间 -> Notebook，然后点击右上角创建 Notebook



步骤 5 名称自定义，然后镜像选择 “pytorch_2.1.0-cuda_12.1-py_3.10.6-ubuntu_22.04-x86_64”，实例规格选择 “CPU: 2 核 8GB”，磁盘规格默认

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
2 PyTorch 编译优化	8
2.1 实验介绍	8
2.1.1 关于本实验	8
2.1.2 实验目的	8
2.2 jit	8
2.2.1 trace 模式	8
2.2.2 script 模式	11
2.3 compile	13
2.4 思考题	14
2.5 本章总结	15

2 PyTorch 编译优化

2.1 实验介绍

2.1.1 关于本实验

本实验包含了 PyTorch 中 jit 模块和 compile 模块的使用，帮助学员了解 PyTorch 编译优化的过程。

2.1.2 实验目的

- 掌握 PyTorch 中 Jit 和 compile 模块的作用。

2.2 jit

torch.jit (Just-In-Time Compiler) 是 PyTorch 的核心模块，主要功能是将 Python 代码或神经网络模型转换为 TorchScript 格式。TorchScript 是 PyTorch 的中间表示 (IR) 。

其中包含了 trace 和 script 两种模式：

Trace 适用无控制流的静态图模型（如固定结构的模型）。

Script 适用含动态控制流的模型（如 RNN、条件分支）。

2.2.1 trace 模式

步骤 1 引入相关依赖的包

输入：

```
import torch
import torch.nn as nn
import torch.jit
import time
```

步骤 2 构建网络结构

此处结构以 LeNet5 结构为例。

输入：

```
# 定义 LeNet 网络结构
```

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5) # 输入 1 通道, 输出 6 通道, 5x5 卷积核
        self.conv2 = nn.Conv2d(6, 16, 5) # 输入 6 通道, 输出 16 通道, 5x5 卷积核
        self.fc1 = nn.Linear(16*61*61, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.pool = nn.AvgPool2d(2, 2) # 2x2 池化窗口
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 16*61*61)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

步骤 3 构建测试函数

输入:

```
# 性能对比测试函数
def benchmark(model, input, iterations=100):
    # 预热
    for _ in range(20):
        _ = model(input)

    # 计时
    start = time.perf_counter()
    for _ in range(iterations):
        _ = model(input)
    end = time.perf_counter()

    return (end - start) / iterations * 1000 # 返回单次推理时间 ( ms )
```

步骤 4 实例化原始模型对象, 并测试推理时间

输入:

```
# 创建模型实例
model = LeNet()

dummy_input = torch.rand(10, 3, 256, 256)
```

```
original_time = benchmark(model, dummy_input)

# 结果输出
print(f"[原始 PyTorch 模型] 平均推理时间: {original_time:.4f} ms")
```

输出:

```
[原始 PyTorch 模型] 平均推理时间: 281.0481 ms
```

```
12 print(f"[原始PyTorch模型] 平均推理时间: {original_time:.4f} ms")
[原始PyTorch模型] 平均推理时间: 281.0481 ms
```

步骤 5 Jit (trace) 格式下推理测试

输入:

```
# 创建模型实例并转换为 Jit 格式
model = LeNet()
# 使用跟踪模式 ( Tracing ) 编译模型
dummy_input = torch.rand(10, 3, 256, 256)
traced_model = torch.jit.trace(model, dummy_input) # 也可以使用装饰器的方式进行调用
traced_model.save("traced_model.pt") ## 保存为 TorchScript 格式方便后续加载, 此处可删除

traced_model = torch.load("traced_model.pt")

jit_time = benchmark(traced_model, dummy_input)

print(f"[Jit 优化模型] 平均推理时间: {jit_time:.4f} ms")
print(f"加速比: {original_time / jit_time:.2f}x")
```

输出:

```
[Jit 优化模型] 平均推理时间: 118.9538 ms
加速比: 2.36x
```

```
[JIT优化模型] 平均推理时间: 118.9538 ms
加速比: 2.36x
```

验证两种模型的输出是否一致。

输入:

```
# 验证输出一致性
with torch.inference_mode():
    original_output = model(dummy_input)
    jit_output = traced_model(dummy_input)

print("\n 输出一致性验证:")
print(f"最大绝对误差: {(original_output - jit_output).abs().max().item():.4e}")
print(f"相对误差: {((original_output - jit_output).abs() / original_output.abs()).max().item():.4e}")
```

输出：

```
输出一致性验证:
最大绝对误差: 0.0000e+00
相对误差: 0.0000e+00
```

```
输出一致性验证:
最大绝对误差: 0.0000e+00
相对误差: 0.0000e+00
```

(此处测试平台为华为云 ModelArts，不同环境和模型结构下测试数据会存在差异。)

2.2.2 script 模式

torch.jit.script 的核心优化在于 消除 Python 解释器开销，直接编译为高效的中间表示 (TorchScript IR) 。

步骤 1 script 代码编译

输入：

```
import torch

# 定义含条件分支的函数
@torch.jit.script
def select_max(x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
    if x.max() > y.max():
        return x
    else:
        return y

# 验证编译结果
print(type(select_max)) # 输出: <class 'torch.jit.ScriptFunction'>
print(select_max.code)  # 查看编译后的 TorchScript 代码

# 调用脚本化函数
input_a = torch.tensor([1.0, 2.0])
input_b = torch.tensor([3.0, 0.5])
output = select_max(input_a, input_b)
print(output)
```

输出：

```
<class 'torch.jit.ScriptFunction'>
def select_max(x: Tensor,
  y: Tensor) -> Tensor:
  _0 = bool(torch.gt(torch.max(x), torch.max(y)))
  if _0:
    _1 = x
  else:
    _1 = y
```

```
return _1  
  
tensor([3.0000, 0.5000])
```

步骤 2 编译后效果测试

新建网络结构。

输入：

```
import torch  
import timeit  
  
class DynamicModel(torch.nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.fc1 = torch.nn.Linear(100, 50)  
        self.fc2 = torch.nn.Linear(50, 10)  
  
    def forward(self, x: torch.Tensor) -> torch.Tensor:  
        # 动态控制流：根据输入特征选择不同路径  
        if x.sum() > 0:  
            h = torch.relu(self.fc1(x))  
        else:  
            h = torch.tanh(self.fc1(x))  
        return self.fc2(h)
```

脚本化推理模型，并定义推理函数。

输入：

```
# 原始模型推理  
def python_inference(model, input_data):  
    return model(input_data)  
  
# 脚本化模型推理  
scripted_model = torch.jit.script(DynamicModel())  
def script_inference(model, input_data):  
    return model(input_data)
```

推理效果比对。

输入：

```
# 输入数据  
input_data = torch.randn(100) # 输入维度为 (100,)  
  
# 测量推理时间（单位：毫秒）  
num_runs = 1000  
  
# 原始模型  
py_time = timeit.timeit(  

```

```

        lambda: python_inference(DynamicModel(), input_data),
        number=num_runs
    ) * 1000 / num_runs

# 脚本化模型
ts_time = timeit.timeit(
    lambda: script_inference(scripted_model, input_data),
    number=num_runs
) * 1000 / num_runs

print(f"原始模型平均推理时间: {py_time:.4f} ms")
print(f"脚本化模型平均推理时间: {ts_time:.4f} ms")
print(f"加速比: {py_time / ts_time:.2f}x")

```

输出:

```

原始模型平均推理时间: 0.2433 ms
脚本化模型平均推理时间: 0.0407 ms
加速比: 5.98x

```

```

原始模型平均推理时间: 0.2433 ms
脚本化模型平均推理时间: 0.0407 ms
加速比: 5.98x

```

2.3 compile

```

import torch
import torch.nn as nn
import timeit

import torch._dynamo
torch._dynamo.config.suppress_errors = True

# 定义简单模型
class SimpleModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(16*30*30, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = torch.flatten(x, 1)
        return self.fc(x)

```

```
# 原始模型推理
def python_inference(model, input_data):
    return model(input_data)

# 编译模型
compiled_model = torch.compile(SimpleModel(), backend='eager')

# 输入数据
input_data = torch.randn(1, 3, 32, 32) # Batch Size=1, Channels=3, H=32, W=32

# 测量推理时间（单位：毫秒）
num_runs = 1000

# 原始模型
py_time = timeit.timeit(
    lambda: python_inference(SimpleModel(), input_data),
    number=num_runs
) * 1000 / num_runs

# 编译后模型
compiled_time = timeit.timeit(
    lambda: compiled_model(input_data),
    number=num_runs
) * 1000 / num_runs

print(f"原始模型平均推理时间: {py_time:.4f} ms")
print(f"编译后模型平均推理时间: {compiled_time:.4f} ms")
print(f"加速比: {py_time / compiled_time:.2f}x")
```

输出：

```
原始模型平均推理时间: 40.8136 ms
编译后模型平均推理时间: 17.9009 ms
加速比: 2.28x
```

2.4 思考题

当模型中存在动态条件分支时，该如何优化？

2.5 本章总结

本章主要介绍 PyTorch 通过 `torch.jit` 和 `torch.compile` 模块实现编译优化加速。

华为认证 AI 系列教程

HCIP-AI-Model Developer

数据工程

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

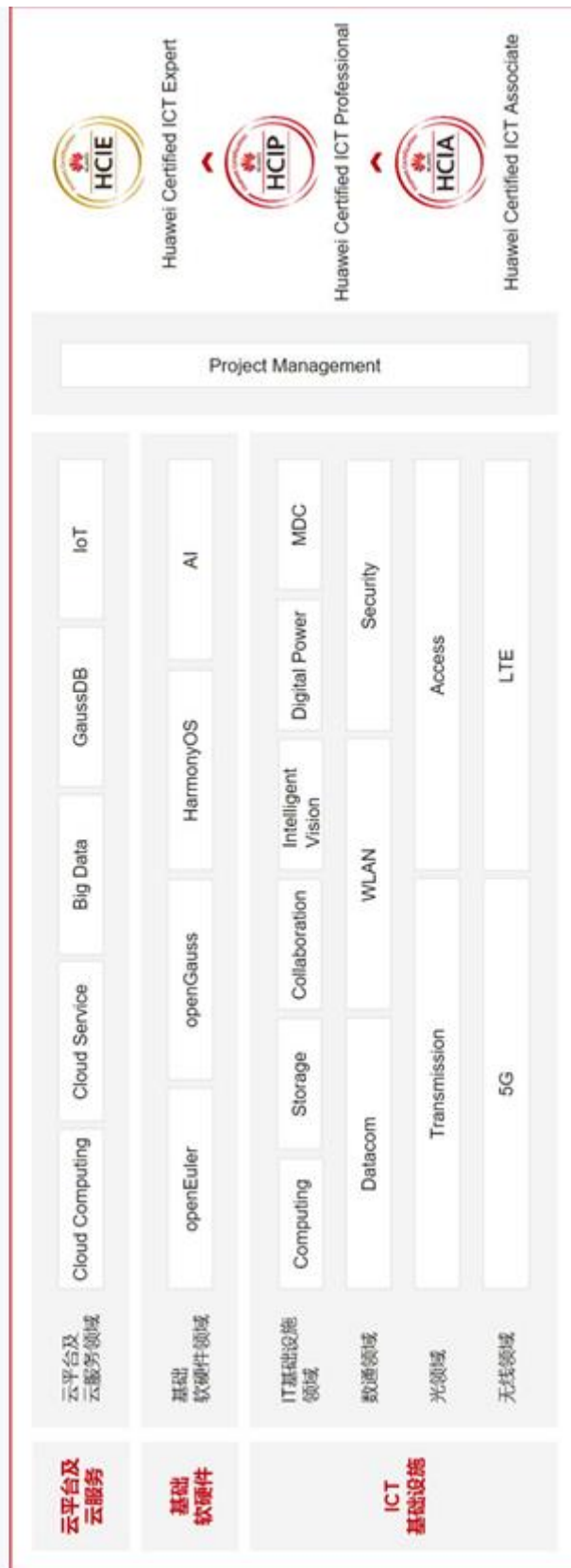
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



前言

简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

本实验指导书共包含 2 个实验，从文本分词开始，逐一介绍了分词器创建、文本处理、模型 API 调用、数据蒸馏等内容。

- 实验一为文本分词实验，使用 DeepSeek 词表文件对文本进行处理，完成文本到 token 以及 token 到文本的转换；
- 实验二为数据蒸馏实验，利用 ModelArts Studio 服务提供的 API 和开源数据集中的问题得到 DeepSeek-R1 的【问题-答案】数据集。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python、AI、LLM 方面知识。

实验环境说明

设备介绍

本实验对硬件无特殊要求，软件要求 Python3.10 及以上，LLM API 要求提供 DeepSeek-R1 模型，以华为云 ModelArts 服务为例：

型号与版本的对应关系如下：

环境名称	实例规格	镜像名称
ModelArts	CPU: 2核 8GB	pytorch_2.1.0-cuda_12.1-py_3.10.6-ubuntu_22.04-x86_64

ModelArts Studio	/	/
------------------	---	---

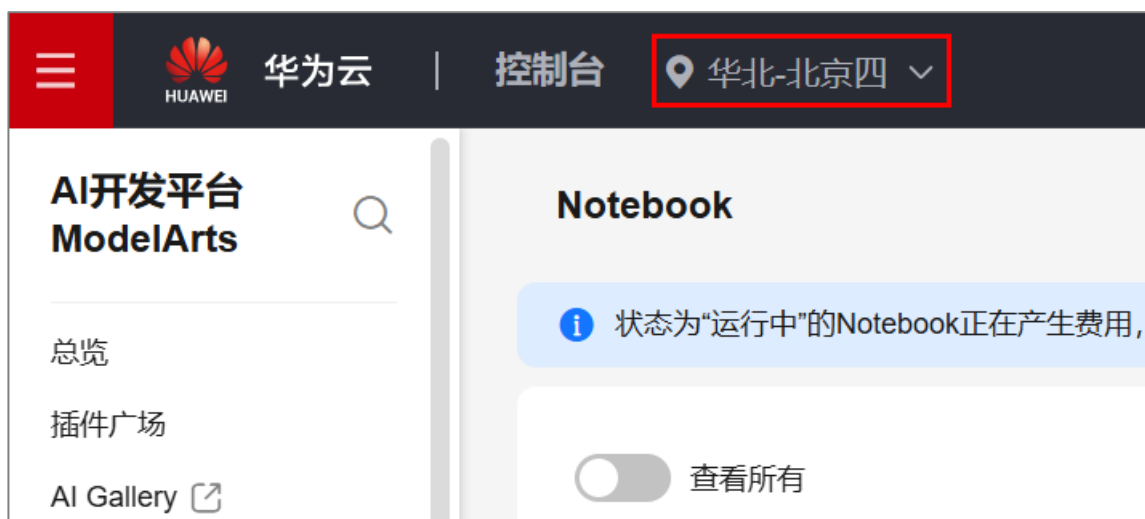
准备实验环境

步骤 1 点击以下链接进入 ModelArts 服务，然后点击控制台进入到后台管理界面

<https://www.huaweicloud.com/product/modelarts.html>



步骤 2 点击左上角切换到北京四区域



步骤 3 点击左侧开发空间 -> Notebook，然后点击右上角创建 Notebook



步骤 4 名称自定义，然后镜像选择“pytorch_2.1.0-cuda_12.1-py_3.10.6-ubuntu_22.04-x86_64”，实例规格选择“CPU: 2 核 8GB”，磁盘规格默认

目录

前 言	4
简介	4
内容描述	4
读者知识背景	4
实验环境说明	4
准备实验环境	5
1 实验环境介绍	8
1.1 实验介绍	8
1.1.1 关于本实验	8
1.1.2 实验设备介绍	8
2 文本分词	9
2.1 实验介绍	9
2.1.1 关于本实验	9
2.1.2 实验目的	9
2.1.3 实验规划	9
2.2 实验任务配置	9
2.2.1 配置步骤	9
2.3 思考题	12
3 数据蒸馏	13
3.1 实验介绍	13
3.1.1 关于本实验	13
3.1.2 实验目的	13
3.1.3 实验规划	13
3.2 实验任务配置	13
3.2.1 配置步骤	13
3.3 思考题	22

1 实验环境介绍

1.1 实验介绍

1.1.1 关于本实验

本实验基于华为云 ModelArts 进行，如果使用其他平台或物理服务器，只要软件规格和设备型号满足要求，也可以进行本实验，实验分为两大部分，文本分词、数据蒸馏。

实验完整代码可点击以下链接下载：

https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI-Model%20Developer/V2.0/data_engineering.ipynb

1.1.2 实验设备介绍

设备型号与软件版本的对应关系如下：

设备型号	软件名称	软件版本
2 核 8G 以上	Python	3.10 以上
/	MaaS	DeepSeek-R1

2 文本分词

2.1 实验介绍

2.1.1 关于本实验

本实验共包含三部分，分别是下载词表文件、加载词表文件、文本分词操作。

2.1.2 实验目的

- 掌握如何利用 transformers 库加载词表文件
- 掌握如何进行文本分词

2.1.3 实验规划

下载词表文件 -> 加载词表文件 -> 完成文本分词。

2.2 实验任务配置

2.2.1 配置步骤

步骤 1 进入 Notebook 界面之后，任意选择一个内核创建 Notebook 即可

步骤 2 输入以下代码创建文件夹并下载 DeepSeek-R1 的词表文件等

```
#创建文件夹，用于存放词表文件等
!mkdir deepseek
#下载词表文件、模型配置文件等
!wget https://hf-mirror.com/deepseek-ai/DeepSeek-R1-0528/resolve/main/config.json?download=true -O ./deepseek/config.json
!wget https://hf-mirror.com/deepseek-ai/DeepSeek-R1-0528/resolve/main/tokenizer_config.json?download=true -O ./deepseek/tokenizer_config.json
!wget https://hf-mirror.com/deepseek-ai/DeepSeek-R1-0528/resolve/main/tokenizer.json?download=true -O ./deepseek/tokenizer.json
!wget https://hf-mirror.com/deepseek-ai/DeepSeek-R1-0528/resolve/main/modeling_deepseek.py?download=true -O ./deepseek/modeling_deepseek.py
!wget https://hf-mirror.com/deepseek-ai/DeepSeek-R1-0528/resolve/main/configuration_deepseek.py?download=true -O ./deepseek/configuration_deepseek.py
```

步骤 3 输入以下代码安装 transformers 依赖库，用于加载词表文件和分词

```
#安装依赖，用于后续分词
!pip install transformers==4.46.3
```

步骤 4 输入以下代码导入依赖库

```
#导入依赖库
from transformers import AutoTokenizer
```

步骤 5 输入以下代码创建分词器

```
#创建分词器
tokenizer = AutoTokenizer.from_pretrained("./deepseek/",trust_remote_code=True)
```

步骤 6 输入以下代码获取词表并展示词元

```
# 获取词表（字典形式：文本 → Token ID）
vocab = tokenizer.get_vocab()

# 打印前 20 个词元及其 ID
for token, idx in list(vocab.items())[:20]:
    print(f"{token}: {idx}")

# 搜索特定词元（如英文"hello"）
print("\n'hello'的 Token ID:", vocab.get("hello", "未找到"))
```

```

ãl-ïä%gä,l: 99882
ĠDNA: 8661
äºIèĠh: 38492
à¹Ġà,gà,@: 91400
Ġproficiency: 45418
clair: 58082
ĠSadd: 93752
qui: 53855
ÃÑdo: 72247
ĠTian: 52608
ÙĠÙĠÙĠÙĠ: 61819
Ġdiscontent: 96674
eta: 5568
Ġrepetition: 42039
ĠAlta: 102552
onometric: 31838
é«Íä,Ġ: 50465
Ġterminal: 16170
èĠIâIĠçlĠ: 47181
æIçlĠ: 53090

'hello'的Token ID: 33310

```

步骤 7 输入以下代码用分词器处理文本，包括文本 -> token，token -> token id，token id -> 文本的转换

```

# 使用分词器处理文本
text = "hello, how are you?"
tokenized_text = tokenizer.tokenize(text)
indexed_tokens1 = tokenizer.convert_tokens_to_ids(tokenized_text)
id_to_tokens = tokenizer.convert_ids_to_tokens(indexed_tokens1)
print('indexed_tokens1:', indexed_tokens1)
print("id_to_tokens:", id_to_tokens)
decoded_text = tokenizer.decode(indexed_tokens1)
print("\n 解码回文本:", decoded_text)

```

```
indexed_tokens1: [33310, 303, 9357, 477, 440, 33]
id_to_tokens: ['hello', 'i', 'how', 'are', 'you', '?']
```

解码回文本: hello, how are you?

步骤 8 输入以下代码直接将文本转换为模型输入格式

```
# 或者直接编码为模型输入的格式
encoded_text = tokenizer.encode(text)
print(encoded_text)
```

```
[0, 33310, 303, 9357, 477, 440, 33]
```

步骤 9 输入以下代码将文本转换为包含 attention_mask 的输入格式

```
# 对于更复杂的输入准备, 可以使用 encode_plus
encoded_plus = tokenizer.encode_plus(text, add_special_tokens=True)
print(encoded_plus)
```

```
{'input_ids': [0, 33310, 303, 9357, 477, 440, 33], 'attention_mask': [1, 1, 1, 1, 1, 1, 1]}
```

2.3 思考题

将文本转换为模型输入的格式与文本 -> token -> token id 有什么区别?

答案: 模型输入格式会在前面增加表示开始的 token, token id 为 0。

3 数据蒸馏

3.1 实验介绍

3.1.1 关于本实验

本实验共包含三部分，分别是 ModelArts Studio 服务开通、LLM API 调用测试、基于 DeepSeek-R1 的数据蒸馏。

3.1.2 实验目的

- 掌握如何调用 LLM API
- 掌握如何编写代码进行数据蒸馏

3.1.3 实验规划

开通 ModelArts Studio 服务 -> LLM API 调用测试 -> 数据蒸馏。

3.2 实验任务配置

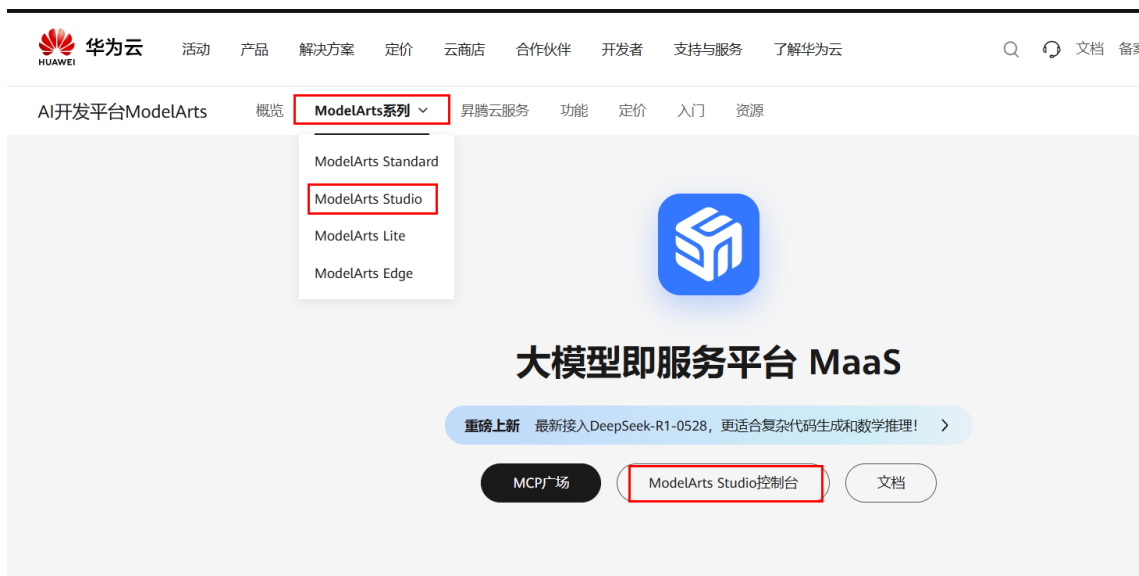
3.2.1 配置步骤

3.2.1.1 ModelArts Studio 服务开通

步骤 1 点击以下链接进入 ModelArts Studio 界面

<https://www.huaweicloud.com/product/modelarts/studio.html>

步骤 2 点击 ModelArts Studio 控制台进入后台界面



步骤 3 选择左侧在线推理，免费服务，点击领取额度（也可选择商用服务，点击开通服务，稳定性更高，但需收费，实验使用的为 DeepSeek-R1）



步骤 4 点击以下链接，根据提示创建 API Key，由于 API Key 只显示一次，建议复制后保存在文件中

<https://console.huaweicloud.com/modelarts/?locale=zh-cn®ion=cn-southwest-2#/model-studio/authmanage>

步骤 5 点击 DeepSeek-R1 右侧的调用说明，获取接口信息

Rest API
OpenAI SDK

接口信息

API地址

https://maas-cn-southwest-2.modelarts-maas.com/v1/infers/8a062fd4-7367-4ab4-a936-5eeb8fb821c4/v1

model参数 ?

DeepSeek-R1

3.2.1.2 模型调用测试

步骤 1 返回 Notebook 界面

步骤 2 输入以下代码安装相关依赖

```
#安装依赖
!pip install openai
!pip install pandas==2.2.3
!pip install numpy==1.26.4
!pip install datasets==3.1.0
!pip install huggingface_hub
```

步骤 3 输入以下代码使用 openai 库测试模型 API 是否可用，base_url 和 api_key 替换为上一步获取的接口信息，包括模型名称，使用其他供应商 API 也可以

```
import openai

client = openai.OpenAI(
    base_url="https://maas-cn-southwest-2.modelarts-maas.com/v1/infers/8a062fd4-7367-4ab4-a936-5eeb8fb821c4/v1", # 替换为供应商 URL
    api_key = "xxx" #替换为自己 API KEY
)

completion = client.chat.completions.create(
    model="DeepSeek-R1", #替换为供应商的模型名称
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "What is the coefficient of  $x^2y^6$  in the expansion of  $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$ ? Express your answer as a common fraction."}
    ],
    stream=True # 启用流式输出
)

def has_valid_index(obj, index):
    """
```

```

检查对象 obj 是否包含指定的 index (角标)
返回 True 如果 obj[index] 有效, 否则返回 False
"""
try:
    _ = obj[index] # 尝试访问索引
    return True
except (IndexError, KeyError, TypeError):
    # IndexError: list/tuple/str 索引越界
    # KeyError: dict 的 key 不存在
    # TypeError: 对象不支持索引访问 (如 int, float 等)
    return False

#用于存储模型输出
reasoning_content = ""
content = ""

#流式输出时会不断返回文本块, 在流式打印的同时将内存存储在列表中
for chunk in completion:
    if has_valid_index(chunk.choices,0):
        if hasattr(chunk.choices[0].delta, 'reasoning_content'):
            reasoning_content += chunk.choices[0].delta.reasoning_content
            print(chunk.choices[0].delta.reasoning_content, end="", flush=True) #打印模型思考内容, 非
reasoning model 则没有
            if chunk.choices[0].delta.content is not None:
                content += chunk.choices[0].delta.content
                print(chunk.choices[0].delta.content, end="", flush=True)
        else:
            print('finish')

```

```

Okay, so I need to find the coefficient of the term  $x^2y^6$  in the expansion of  $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$ . Hmm, let's see. I remember that when expanding binomials raised to a power, we can use the binomial theorem. The binomial theorem states that  $(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$ .

In this case, the binomial is  $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$ , so I can think of  $(a)$  as  $\frac{3}{5}x$  and  $(b)$  as  $-\frac{y}{2}$ . The exponent  $(n)$  is 8.

The general term in the expansion will be  $\binom{8}{k} \left(\frac{3}{5}x\right)^{8-k} \left(-\frac{y}{2}\right)^k$ . I need to find the term where the power of  $(x)$  is 2 and the power of  $(y)$  is 6.

Since the term has  $x^2y^6$ , that means in the general term,  $\left(\frac{3}{5}x\right)^{8-k}$  should have  $x^2$  and  $\left(-\frac{y}{2}\right)^k$  should have  $y^6$ . Therefore, the exponents must satisfy:

For the  $(x)$  term:  $8 - k = 2$ 
For the  $(y)$  term:  $k = 6$ 

Let me check that. If  $(8 - k = 2)$ , then solving for  $(k)$  gives  $(k = 6)$ . That's consistent with the  $(y)$  term having exponent  $(k = 6)$ . Perfect, so the term we're interested in corresponds to  $(k = 6)$ .

Now, let's compute the coefficient for this term. The coefficient is given by:


$$\binom{8}{6} \left(\frac{3}{5}\right)^{8-6} \left(-\frac{1}{2}\right)^6$$


First, compute the binomial coefficient  $\binom{8}{6}$ . I remember that  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ . So,

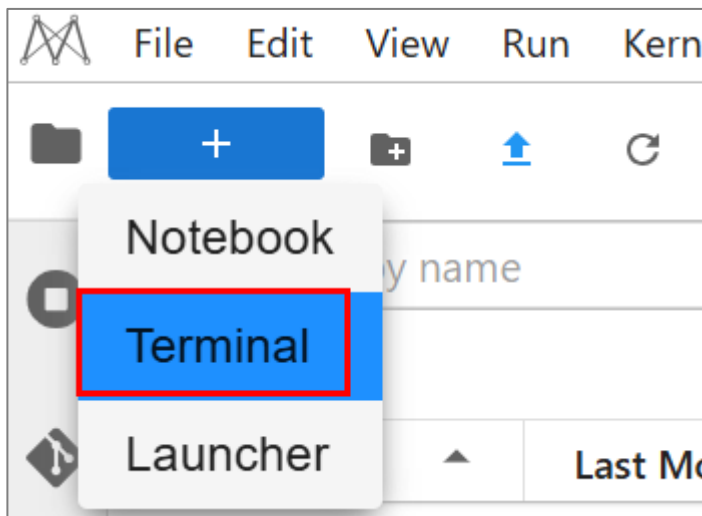
```

3.2.1.3 基于 DeepSeek-R1 蒸馏带有思考过程的数据集

模型 API 测试可用之后, 就可以使用以下代码蒸馏数据集了。

步骤 1 在 Notebook 中输入以下代码下载开源数据集 (本实验仅使用该数据集的问题, 答案由 DeepSeek-R1 生成)

步骤 2 点击左上角，新建 Terminal，用于下载样例数据集



步骤 3 输入以下命令下载样例数据集（该数据集从 NuminaMath-CoT 数据集中选择了大约 70k 个问题，重点关注具有数值输出的问题，其中大多数是整数，本实验利用了该数据集的问题，并未使用答案。）

```
pip install huggingface_hub
export HF_ENDPOINT=https://hf-mirror.com
hf download --repo-type dataset AI-MO/NuminaMath-TIR --local-dir ./AI-MO/NuminaMath-TIR
```

```
Fetching 4 files: 50%|██████████| 2/4 [00:01<00:01, 1.90it/s]
Downloading 'data/train-00000-of-00001.parquet' to 'AI-MO/NuminaMath-TIR/.cache/huggingface/download/data/jhdVX7QxSai
5xYQbu6FpvD3SSxM=.66163f9fdf2220fed2116142781fe7019a117191e7f08a3339a70a98401061d8.incomplete'
test-00000-of-00001.parquet: 100%|██████████| 215k/215k [00:00<00:00, 413kB/s]
Download complete. Moving file to AI-MO/NuminaMath-TIR/data/test-00000-of-00001.parquet
train-00000-of-00001.parquet: 100%|██████████| 147M/147M [00:09<00:00, 15.6MB/s]
Download complete. Moving file to AI-MO/NuminaMath-TIR/data/train-00000-of-00001.parquet
Fetching 4 files: 100%|██████████| 4/4 [00:12<00:00, 3.03s/it]
/home/ma-user/work/AI-MO/NuminaMath-TIR
```

步骤 4 返回 Notebook，输入以下代码导入依赖库（如果报错则重启 Kernel，重新执行代码）

```
import os
from datasets import load_dataset
```

步骤 5 输入以下代码设置提示词模板并加载数据集（下面代码中也包含提示词模板，这里只是作为展示）

```
#设置提示词模板，让模型回答更容易带有推理过程的答案
prompt_template = """\
You will be given a problem. Please reason step by step, and put your final answer within \boxed{:
{{ instruction }}"""
#加载数据集
dataset = load_dataset("./AI-MO/NuminaMath-TIR", split="train").select(range(10))
```

步骤 6 输入以下代码查看第一个问题

```
#查看数据集的第一个问题
```

```
dataset['problem'][0]
```

```
'What is the coefficient of  $x^2y^6$  in the expansion of  $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$ ? Express your answer as a common fraction.'
```

步骤 7 输入以下代码设置模型蒸馏主体函数，具体每个函数功能见注释

```
import json
import os
from typing import List, Dict, Optional
import openai # 或其他 LLM 的 SDK

class LLMBatchProcessor:
    def __init__(self, api_key: str, model_name: str = "gpt-3.5-turbo", base_url: Optional[str] = None):
        """
        初始化 LLM 批量处理器

        :param api_key: LLM API 密钥
        :param model_name: 使用的模型名称，默认为 gpt-3.5-turbo
        :param base_url: 自定义 API 基础 URL，用于本地或第三方部署的 LLM
        """
        self.api_key = api_key
        self.model_name = model_name
        self.base_url = base_url
        self.prompt_template = """\
You will be given a problem. Please reason step by step, and put your final answer within \boxed{ }:
{{ instruction }}"""

        # 初始化 API 客户端
        openai.api_key = self.api_key
        if self.base_url:
            openai.api_base = self.base_url

    def generate_prompt(self, question: str) -> str:
        """
        根据问题生成完整的 prompt

        :param question: 要处理的问题
        :return: 完整的 prompt 字符串
        """
        return self.prompt_template.replace("{{ instruction }}", question)

    def has_valid_index(self, obj, index):
        """
        检查对象 obj 是否包含指定的 index (角标)
        返回 True 如果 obj[index] 有效，否则返回 False
        """
        try:
            _ = obj[index] # 尝试访问索引
```

```
        return True
    except (IndexError, KeyError, TypeError):
        # IndexError: list/tuple/str 索引越界
        # KeyError: dict 的 key 不存在
        # TypeError: 对象不支持索引访问 (如 int, float 等)
        return False

def call_llm_api(self, question: str) -> str:
    """
    调用 LLM API 获取回答

    :param question: 要处理的问题
    :return: 模型的回答
    """
    prompt = self.generate_prompt(question)
    reasoning_content = ""
    content = ""
    if self.base_url:
        client = openai.OpenAI(
            base_url = self.base_url,
            api_key = self.api_key
        )
    else:
        client = openai.OpenAI(
            api_key = self.api_key
        )
    try:
        response = client.chat.completions.create(
            model=self.model_name,
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": prompt}
            ],
            stream=True,
            temperature=0.6
        )

        for chunk in response:
            if self.has_valid_index(chunk.choices,0):
                if hasattr(chunk.choices[0].delta, 'reasoning_content'):
                    reasoning_content += chunk.choices[0].delta.reasoning_content
                    #可设置隐藏输出打印
                    print(chunk.choices[0].delta.reasoning_content, end="",flush=True)
                if chunk.choices[0].delta.content is not None:
                    content += chunk.choices[0].delta.content
                    #可设置隐藏输出打印
                    print(chunk.choices[0].delta.content, end="",flush=True)
```

```
        else:
            print('finish')
            #拼接模型的思考过程和最终输出
            return '/think/'+reasoning_content+'/think/'+content
    except Exception as e:
        print(f"调用 API 时出错: {e}")
        return None

def process_questions(
    self,
    questions: List[str],
    output_file: str,
    batch_size: int = 2,
    start_index: int = 0,
    max_questions: Optional[int] = None
) -> None:
    """
    批量处理问题列表并保存结果

    :param questions: 问题列表
    :param output_file: 输出 JSON 文件路径
    :param batch_size: 每次处理的问题数量，默认为 2
    :param start_index: 开始处理的问题索引，默认为 0
    :param max_questions: 最大处理问题数量，None 表示处理所有
    """
    # 加载已有结果(如果存在)
    existing_results = []
    if os.path.exists(output_file):
        with open(output_file, 'r', encoding='utf-8') as f:
            existing_results = json.load(f)

    # 确定处理范围
    end_index = len(questions)
    if max_questions is not None:
        end_index = min(start_index + max_questions, len(questions))

    # 只处理未处理过的问题
    processed_questions = {result['question'] for result in existing_results}
    new_results = existing_results.copy()

    # 分批处理问题
    for i in range(start_index, end_index, batch_size):
        batch = questions[i:i+batch_size]
        print(f"正在处理问题 {i+1}-{min(i+batch_size, end_index)}/{len(questions)} (共{end_index-start_index}个待处理)")
```

```

for question in batch:
    if question in processed_questions:
        print(f"问题已处理过，跳过: {question[:50]}...")
        continue

    print(f"处理问题: {question[:50]}...")
    print(f"处理问题: {question}...")
    answer = self.call_llm_api(question)

    if answer is not None:
        result = {
            "question": question,
            "answer": answer
        }
        new_results.append(result)
        processed_questions.add(question)

# 保存当前批次结果
self._save_results(new_results, output_file)

def _save_results(self, results: List[Dict], output_file: str) -> None:
    """
    保存结果到 JSON 文件

    :param results: 结果列表
    :param output_file: 输出文件路径
    """
    # 确保目录存在
    os.makedirs(os.path.dirname(output_file), exist_ok=True)

    try:
        with open(output_file, 'w', encoding='utf-8') as f:
            json.dump(results, f, ensure_ascii=False, indent=2)
            print(f"结果已保存到 {output_file} (共{len(results)}条记录)")
    except Exception as e:
        print(f"保存文件时出错: {e}")

```

步骤 8 输入以下代码配置参数，包括 API_KEY、BASE_URL、MODEL_NAME，同时启动数据蒸馏，代码设置每次处理 2 个问题，最多处理 4 个，可根据实际需求调整

```

# 配置参数
API_KEY = "xxx" # 替换为自己 API KEY
BASE_URL = "https://maas-cn-southwest-2.modelarts-maas.com/v1/infers/8a062fd4-7367-4ab4-a936-5eeb8fb821c4/v1" # 替换为供应商 URL
MODEL_NAME="DeepSeek-R1"#替换为供应商模型名称
OUTPUT_FILE = "results/llm_answers.json" # 蒸馏数据集输出路径
QUESTIONS = dataset['problem']

```

```
# 创建处理器实例
processor = LLMBatchProcessor(
    api_key=API_KEY,
    base_url=BASE_URL, # 如果不使用自定义 URL, 可以省略此参数
    model_name=MODEL_NAME
)

# 处理问题并保存结果(每次处理 2 个, 最多处理 4 个问题)
processor.process_questions(
    questions=QUESTIONS,
    output_file=OUTPUT_FILE,
    batch_size=2, # 每批处理 2 个问题
    max_questions=4 # 最多处理 4 个问题
)
```

正在处理问题 1-2/10 (共4个待处理)

处理问题: What is the coefficient of x^2y^6 in the expansion of $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$? Express your answer as a common fraction....

处理问题: What is the coefficient of x^2y^6 in the expansion of $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$? Express your answer as a common fraction....

Okay, so I need to find the coefficient of the term x^2y^6 in the expansion of $\left(\frac{3}{5}x - \frac{y}{2}\right)^8$. Hmm, let's see. I remember that when expanding binomials raised to a power, we can use the binomial theorem. The binomial theorem states that $(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$.

In this case, the binomial is $\left(\frac{3}{5}x - \frac{y}{2}\right)$, so a is $\frac{3}{5}x$ and b is $-\frac{y}{2}$. The exponent n is 8. So, each term in the expansion will be of the form $\binom{8}{k} \left(\frac{3}{5}x\right)^{8-k} \left(-\frac{y}{2}\right)^k$.

Our goal is to find the term where the power of x is 2 and the power of y is 6. Let me think. Since in each term, the exponents of x and y come from the exponents in a and b . Specifically, the exponent of x is $(8 - k)$ because a is raised to $(8 - k)$ and a has an x term. Similarly, the exponent of y is k because b is raised to the k th power and b has a y term.

We need the exponent of x to be 2 and the exponent of y to be 6. Therefore, we can set up the equations:

$8 - k = 2$ (for x 's exponent)

$k = 6$ (for y 's exponent)

So, solving $8 - k = 2$ gives $k = 6$. That checks out, so the term when $k=6$ will have x^2 and y^6 .

3.3 思考题

构造自定义数据集, 然后修改代码对 DeepSeek-R1 进行数据蒸馏。

思路: 先制作自定义数据集, 可以用列表, 然后蒸馏的问题来源指向改为自定义数据集即可。

华为认证 AI 系列教程

HCIP-AI-Model Developer

模型迁移与调优

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

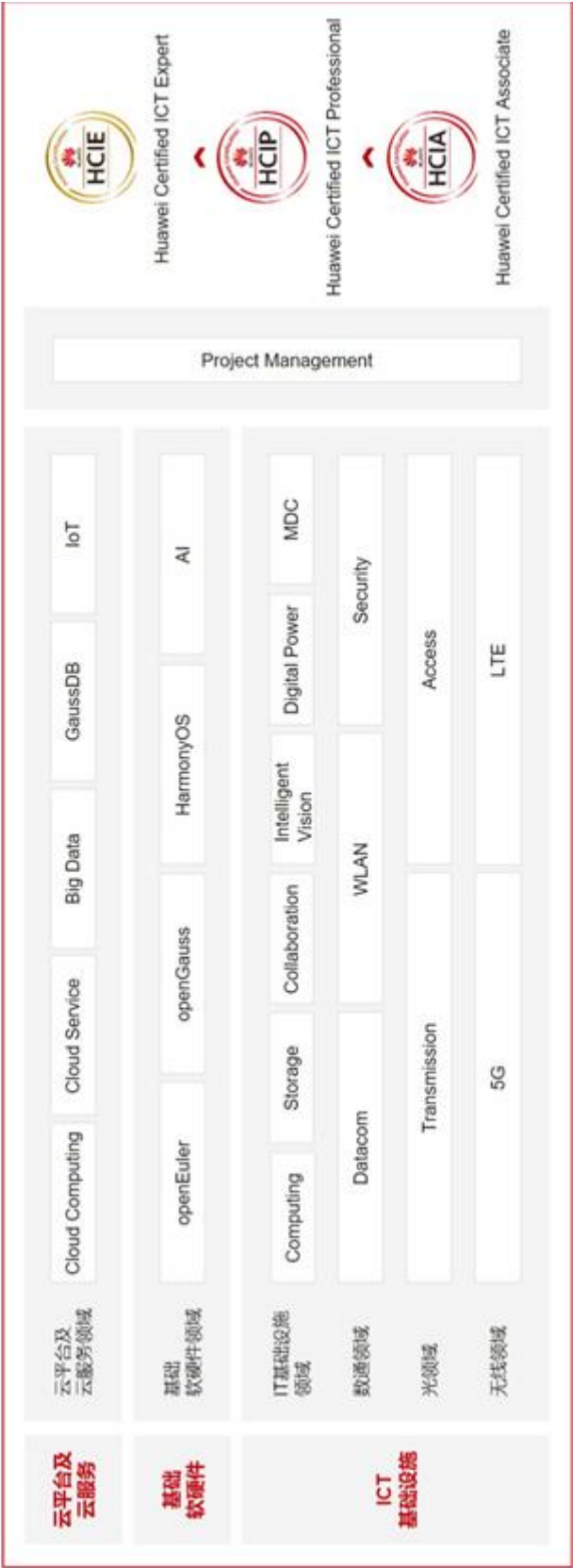
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

本实验指导书共包含 3 个实验，从模型构建开始，依次介绍了在 CPU 上执行模型训练、将训练迁移到 NPU、在 NPU 上进行性能调优。

- 实验一主要讲述了如何基于 PyTorch 构建一个简单的卷积神经网络，包含网络构建、优化器定义、模型训练等。
- 实验二主要讲述了如何将实验一的模型训练迁移到 NPU 上执行，包括网络和数据集的迁移。
- 实验三主要讲述了如何收集实验二的性能数据，然后使用性能分析工具给出优化建议。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本 Python、AI 方面知识，同时了解 PyTorch 框架。

实验环境说明

设备介绍

本实验需要使用华为昇腾芯片，并使用配套的 CANN 软件包，为了满足 HCIA-AI-Model Developer 实验需要，建议每套实验环境采用以下配置，以华为云 ModelArts 服务为例：型号与版本的对应关系如下：

环境名称	实例规格	镜像名称
ModelArts	Ascend: 1*ascend-snt9b1 ARM: 24核 192GB	pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b

准备实验环境

步骤 1 点击以下链接进入 ModelArts 服务，然后点击控制台进入到后台管理界面

<https://www.huaweicloud.com/product/modelarts.html>



步骤 2 点击左上角切换到贵阳一区域



步骤 3 点击左侧开发空间 -> Notebook，然后点击右上角创建 Notebook



步骤 4 名称自定义，然后镜像选择“pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b”，实例规格选择“Ascend: 1*ascend-snt9b1|ARM: 24 核 192GB”，磁盘规格改为 100GB

< | 创建Notebook

☐ mindspore_2.4.10-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-... Ascend Snt9B+ARM算法开发和训练基础镜像，AI引擎预置MindSpore

☒ pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b Ascend_snt9b+ARM算法开发和训练基础镜像，AI引擎预置PyTorch...

* 资源类型

* 类型

* 实例规格 Ascend: 1*ascend-snt9b1|ARM: 24核 192GB
昇腾规格，适合深度学习场景下的模型训练和调测

* 存储配置 云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留，存储支持在线按需扩容

* 磁盘规格 GB
磁盘规格默认为5GB，从Notebook实例创建成功起，直至删除成功，每GB按照规定费用收费。

步骤 5 创建成功后点击打开即可进入 Notebook 界面

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
准备实验环境	4
2 实验环境介绍	8
2.1 实验介绍	8
2.1.1 关于本实验	8
2.1.2 实验设备介绍	8
3 模型构建	9
3.1 实验介绍	9
3.1.1 关于本实验	9
3.1.2 实验目的	9
3.1.3 实验规划	9
3.2 实验任务配置	9
3.2.1 配置步骤	9
3.3 思考题	15
4 模型迁移	16
4.1 实验介绍	16
4.1.1 关于本实验	16
4.1.2 实验目的	16
4.1.3 实验规划	16
4.2 实验任务配置	16
4.2.1 配置步骤	16
4.3 思考题	19
5 模型性能调优	20
5.1 实验介绍	20
5.1.1 关于本实验	20
5.1.2 实验目的	20



5.1.3 实验规划.....	20
5.2 实验任务配置	20
5.2.1 配置步骤.....	20
5.3 思考题	25

1

实验环境介绍

1.1 实验介绍

1.1.1 关于本实验

本实验基于华为云 ModelArts 进行，如果使用其他平台或物理服务器，只要软件规格和设备型号满足要求，也可以进行本实验，实验分为三大部分，模型构建、模型迁移、模型性能调优。

1.1.2 实验设备介绍

设备型号与软件版本的对应关系如下：

设备型号	软件名称	软件版本
Atlas A2 系列训练设备	CANN	8.0.0 及以上

2 模型构建

2.1 实验介绍

2.1.1 关于本实验

本实验共包含五部分，分别是数据集加载、网络定义、损失函数和优化器定义、网络训练、模型测试。

实验完整代码可点击以下链接下载：

https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI-Model%20Developer/V2.0/model_migration%26optimization.ipynb

2.1.2 实验目的

- 掌握如何利用 PyTorch 加载数据集
- 掌握如何利用 PyTorch 定义神经网络
- 掌握如何利用 PyTorch 定义损失函数
- 掌握如何利用 PyTorch 训练模型
- 掌握如何利用 PyTorch 测试模型

2.1.3 实验规划

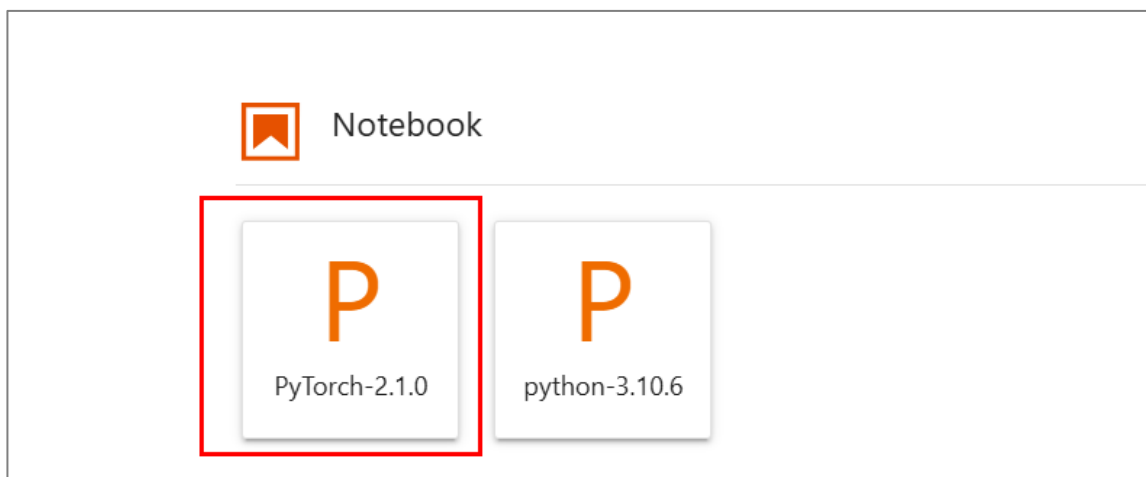
加载 CIFAR10 数据集 -> 定义卷积神经网络 -> 定义损失函数和优化器 -> 训练神经网络 -> 在测试集中测试模型。

2.2 实验任务配置

2.2.1 配置步骤

2.2.1.1 加载 CIFAR10 数据集

步骤 1 进入 Notebook 界面之后，选择 PyTorch-2.1.0 内核创建 Notebook



步骤 2 创建 Notebook 后，输入以下代码，导入相关依赖库

```
import torch
import torchvision
import torchvision.transforms as transforms
```

步骤 3 输入以下代码下载 CIFAR10 数据集，然后利用 torchvision 加载数据集并进行图像标准化，同时展示样本数量

```
#torchvision 加载是图片在[0,1]之间，通过代码转换到[-1,1]
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                         download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                           shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                         download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                          shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

#打印训练数据集样本数量
print(len(trainset))
```

50000

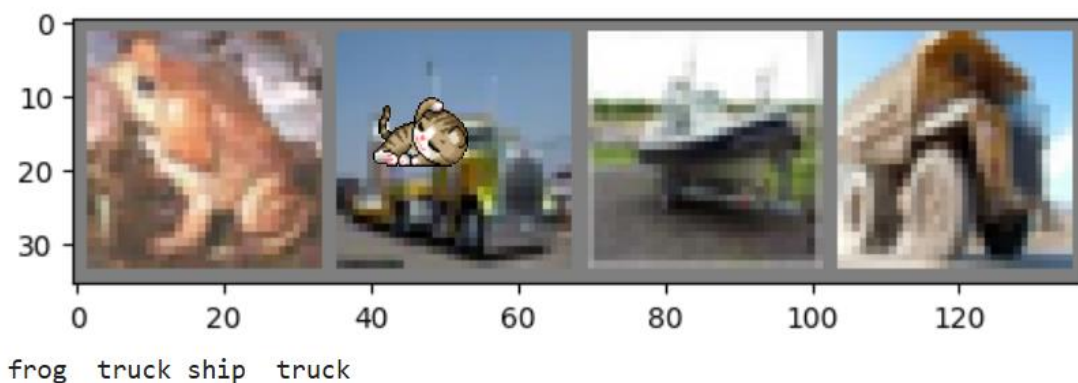
步骤 4 输入以下代码可随机展示部分样本图像

```
import matplotlib.pyplot as plt
import numpy as np

def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# 得到一些随机的训练图像
dataiter = iter(trainloader)
images, labels = next(dataiter)

# 展示图像
imshow(torchvision.utils.make_grid(images))
# 打印标签
print(' '.join('%5s' % classes[labels[j]]:5s for j in range(batch_size)))
```



2.2.1.2 定义卷积神经网络

步骤 1 输入以下代码定义一个简单的卷积神经网络

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = torch.flatten(x, 1) # flatten all dimensions except batch
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x

net = Net()
```

2.2.1.3 定义损失函数和优化器

步骤 1 输入以下代码定义损失函数和优化器

```
#使用交叉熵损失函数和 SGD 优化器
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

2.2.1.4 训练神经网络

步骤 1 现在神经网络、损失函数、优化器、数据集等都已经定义好，输入以下代码迭代获取数据，开始网络训练，共训练 200 小批，每批 4 个样本（见 2.2.2.1 步骤的 batch_size）

```
#循环取得数据，然后作为输入给神经网络进行优化
for epoch in range(2): # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 获取输入；数据是【输入，标签】的列表
        inputs, labels = data

        # 将参数梯度归零
        optimizer.zero_grad()

        # 正向+反向+优化
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # 打印统计数据
        running_loss += loss.item()
        if i % 200 == 199: # 每 200 小批打印一次
```

```
print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 200:.3f}')
running_loss = 0.0

# CPU 上训练较慢，在 200 小批的时候终止训练
if i % 200 == 199:
    break

print('Finished Training')
```

步骤 2 输入以下代码保存上一步训练的模型网络参数，可在加载后用于图像预测

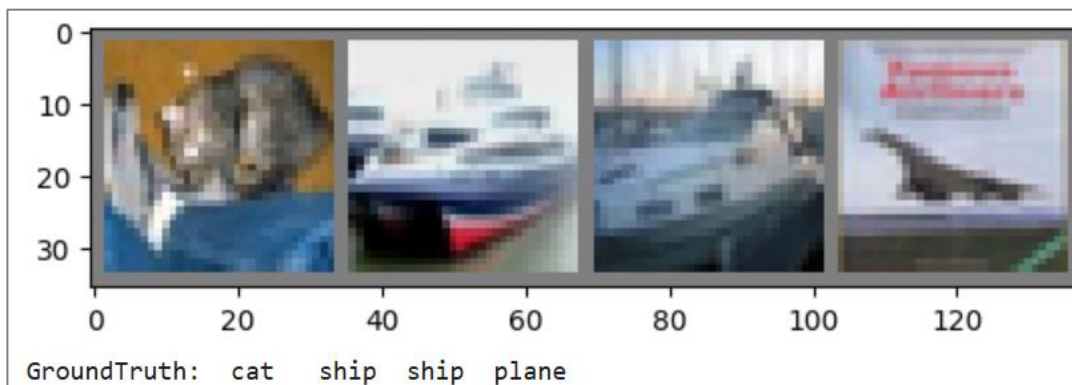
```
#保存模型
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

2.2.1.5 在测试集中测试模型

步骤 1 输入以下代码随机展示测试集中的图像

```
'''
模型训练完成后，让模型预测测试集中样本的标签，然后和真实标签进行比对，以此判断模型性能，下面代
码先展示部分图片
'''
dataiter = iter(testloader)
images, labels = next(dataiter)

# 打印图像
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```



步骤 2 输入以下代码加载前面保存的模型权重参数

```
#加载网络和权重参数
net = Net()
net.load_state_dict(torch.load(PATH, weights_only=True))
```

步骤 3 输入以下代码使用上一步加载的模型对图像进行预测

```
#得到模型预测结果
outputs = net(images)
```

步骤 4 由于模型给出的预测结果是一个包含各个类别的向量，如（0.1，0.2，0.7），因此需要处理得到概率最大的类别作为模型预测结果，输入以下代码展示模型预测类别

```
#模型输出结果是每个类别的概率值，通过下面代码取得概率最高的类别，即模型预测的类别结果
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s'%classes[predicted[j]]
                                for j in range(4)))
```

```
Predicted:   cat   plane plane plane
```

步骤 5 （可选）输入以下代码在整个测试集上进行测试，得到模型预测准确率和每个类别分别的准确率，但由于目前代码在 CPU 上执行，因此速度较慢

```
#在整个测试集上进行测试，得到结果
#由于 CPU 计算较慢，因此这里省略这一部分

correct = 0
total = 0
# 因为我们没有训练，所以我们不需要计算输出的梯度
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # 得到模型预测结果
        outputs = net(images)
        # 模型输出结果是每个类别的概率值，通过下面代码取得概率最高的类别，即模型预测的类别结果
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')

correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

#计算每个类别的精度
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)

        for label, prediction in zip(labels, predictions):
            if label == prediction:
```

```
        correct_pred[classes[label]] += 1
        total_pred[classes[label]] += 1

# 打印每个类的精度
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

2.3 思考题

在上面的操作步骤中，修改哪些参数可以减少模型训练时间？

答案：减少训练 step、减少网络层数等都可以。

3 模型迁移

3.1 实验介绍

3.1.1 关于本实验

本实验共包含五部分，分别是导入 torch_npu 依赖库、转移网络到 NPU、转移数据到 NPU、基于 NPU 训练模型、基于 NPU 测试模型。

3.1.2 实验目的

- 掌握如何用手工迁移的方式将模型迁移到 NPU 上运行

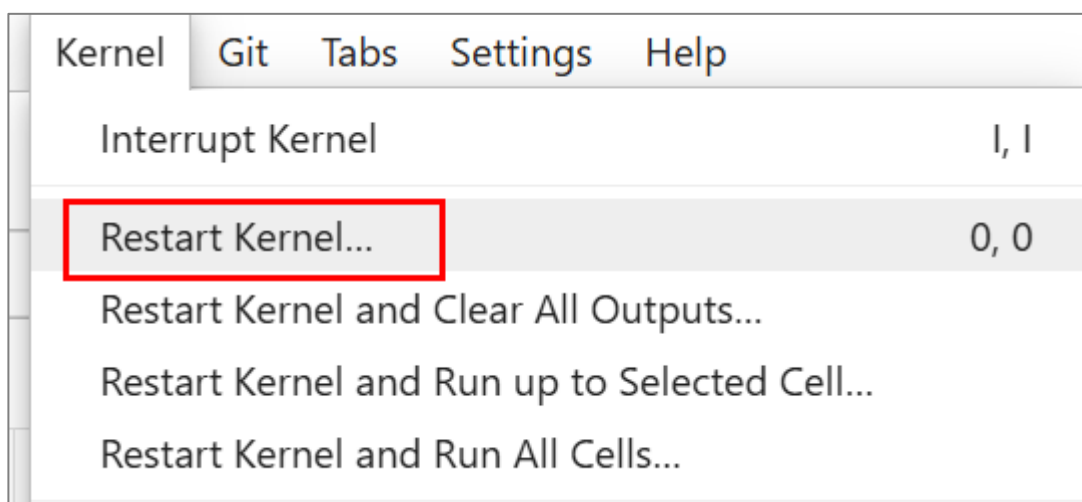
3.1.3 实验规划

添加依赖库 -> 增加 NPU 状态检查代码 -> 将网络迁移到 NPU -> 将数据迁移到 NPU -> 基于 NPU 进行模型训练 -> 基于 NPU 进行模型测试。

3.2 实验任务配置

3.2.1 配置步骤

步骤 1 点击左上角，重启内核，清除上一个实验产生的所有变量



步骤 2 在导入依赖库的部分，增加导入 torch_npu 的代码（在 PyTorch2.5.1 之后无需添加，本实验 PyTorch 版本为 2.1.0，需要添加）

```
#增加 torch_npu，让 torch 可以在 npu 上运行
import torch_npu
```

步骤 3 在加载 CIFAR10 数据集步骤之前添加以下代码，判断 NPU 是否可用（可点击单元格左侧后按 A 键在单元格上方添加空白单元格），由于实验环境为单卡，因此 format 后写 0

```
#判断 NPU 是否可用
device = torch.device('npu:{}'.format(0))
print(device)
```

步骤 4 在定义卷积神经网络部分添加以下代码，将定义好的网络放到 NPU 设备上

```
"""把网络迁移到 NPU 上"""
net.to(device)
```

步骤 5 在训练神经网络部分修改数据获取部分代码，将数据部分也放到 NPU 设备上

```
#获取输入；数据是【输入，标签】的列表，将数据放到 NPU 上
inputs, labels = data[0].to(device), data[1].to(device)
```

步骤 6 在训练神经网络部分修改打印统计数据部分代码（200 变 2000），同时删除 200 小批终止训练的代码

```
# 打印统计数据
running_loss += loss.item()
if i % 2000 == 1999: # 每2000小批打印一次
    print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
    running_loss = 0.0

print('Finished Training')
```

步骤 7 其余代码不变，可重新执行模型训练并测试模型在测试数据集的表现（模型测试时同样需要增加代码将 net 和 inputs 迁移到 NPU），小批次样本预测部分修改代码参考如下

```
#得到模型预测结果
inputs, labels = images.to(device), labels.to(device)
outputs = net(inputs)
```

步骤 8 最终测试代码与上个实验相比仅修改了数据部分，将数据迁移到 NPU，代码如下

```
#在整个测试集上进行测试，得到结果

correct = 0
total = 0
# 因为我们没有训练，所以我们不需要计算输出的梯度
with torch.no_grad():
```

```
for data in testloader:
    images, labels = data[0].to(device), data[1].to(device)
    # 得到模型预测结果
    outputs = net(images)
    # 模型输出结果是每个类别的概率值，通过下面代码取得概率最高的类别，即模型预测的类别结果
    _, predicted = torch.max(outputs, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')

correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

#计算每个类别的预测准确率
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)

        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1

# 打印每个类的精度
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
Accuracy of the network on the 10000 test images: 55 %
Accuracy for class: plane is 62.1 %
Accuracy for class: car   is 73.8 %
Accuracy for class: bird  is 35.6 %
Accuracy for class: cat   is 32.0 %
Accuracy for class: deer  is 40.4 %
Accuracy for class: dog   is 60.5 %
Accuracy for class: frog  is 59.6 %
Accuracy for class: horse is 54.4 %
Accuracy for class: ship  is 77.8 %
Accuracy for class: truck is 62.8 %
```

3.3 思考题

可以通过修改哪些参数提高 NPU 的利用率或者提高模型最终预测精度？

答案：增加 batch size 可提升 NPU 利用率，增加 epoch 和网络层数可以提升最终预测精度。

4 模型性能调优

4.1 实验介绍

4.1.1 关于本实验

本实验共包含两部分，分别是性能数据采集和性能数据分析。

4.1.2 实验目的

- 掌握如何使用 PyTorch 自带模块采集性能数据
- 掌握如何使用性能分析工具分析性能数据

4.1.3 实验规划

性能数据采集 -> 性能数据分析。

4.2 实验任务配置

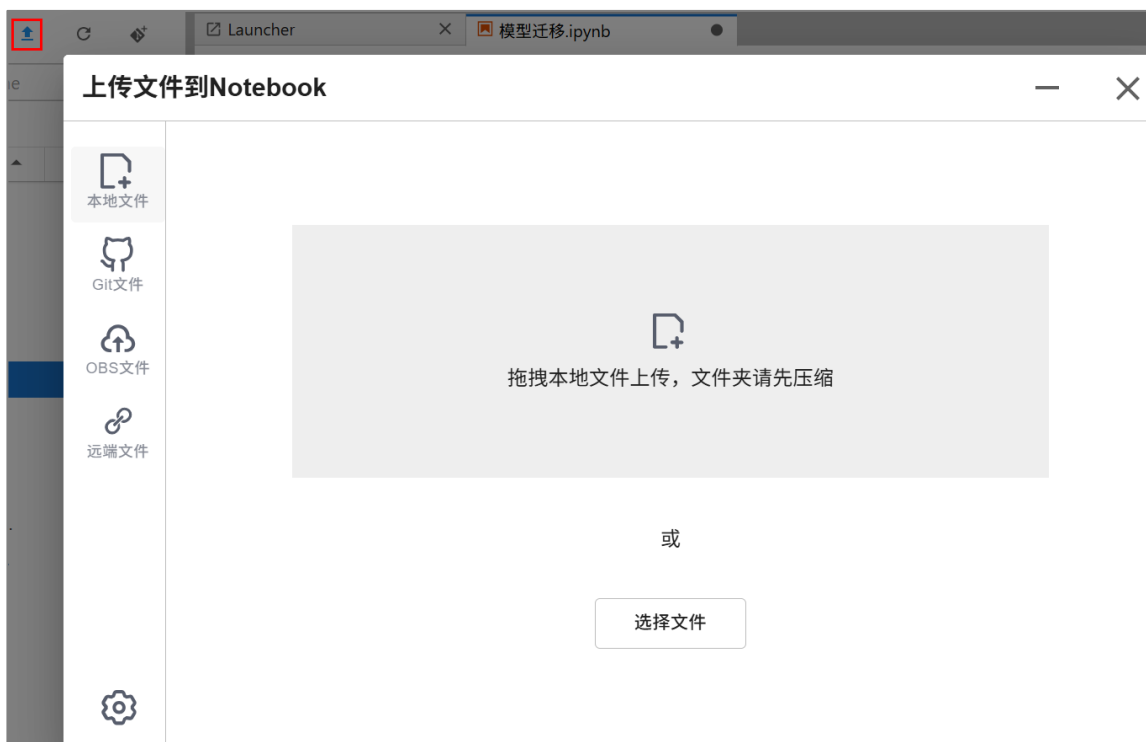
4.2.1 配置步骤

4.2.1.1 性能数据采集

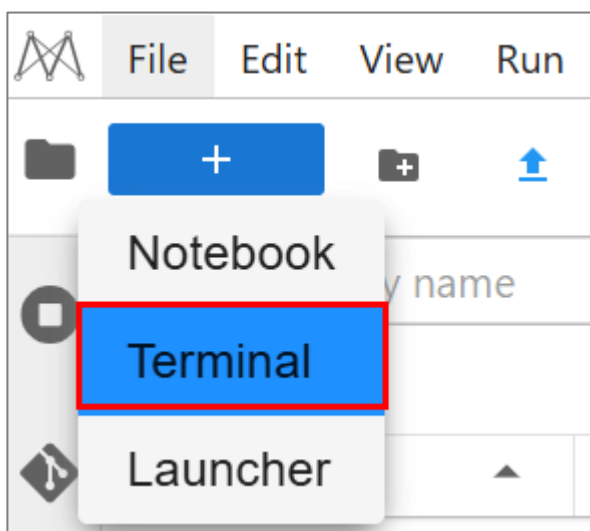
步骤 1 单击以下链接下载性能测试工具安装包（由于 ModelArts 安全策略设置，需要把安装包下载到本地后上传 ModelArts 进行安装）

https://ptdbg.obs.cn-north-4.myhuaweicloud.com/profiler/package/8.1.0/msprof_analyze-8.1.0-py3-none-any.whl

步骤 2 点击左上角，上传安装包到 Notebook



步骤 3 点击左上角，新建一个 Terminal 窗口



步骤 4 确认当前虚拟环境为 PyTorch-2.1.0，如果不是，可输入以下命令切换

```
conda activate PyTorch-2.1.0
```

步骤 5 输入以下命令查看安装包是否存在

```
ls
```

```
(PyTorch-2.1.0) [ma-user work]$ls
cifar_net.pth          fusion_result.json      NPU性能数据采集.ipynb
computation_analysis.ipynb kernel_meta             profiling_data
data                  lost+found              模型迁移.ipynb
fusion_operators_api_analysis.ipynb msprof_analyze-8.1.0-py3-none-any.whl
```

步骤 6 输入以下命令安装工具

```
pip install msprof_analyze-8.1.0-py3-none-any.whl
```

步骤 7 安装完成后返回 Notebook，然后修改模型迁移后的代码中训练神经网络部分，修改后的代码内容如下，可直接覆盖原来代码

```
experimental_config = torch_npu.profiler._ExperimentalConfig(
    aic_metrics=torch_npu.profiler.AiCMetrics.PipeUtilization,
    profiler_level=torch_npu.profiler.ProfilerLevel.Level1,
    l2_cache=False
)
with torch_npu.profiler.profile(
    activities=[
        torch_npu.profiler.ProfilerActivity.CPU,
        torch_npu.profiler.ProfilerActivity.NPU
    ],
    record_shapes=True,
    profile_memory=True,
    with_stack=True,
    experimental_config=experimental_config,
    schedule=torch_npu.profiler.schedule(wait=10, warmup=0, active=1, repeat=1),
    on_trace_ready=torch_npu.profiler.tensorboard_trace_handler("./profiling_data")
) as prof:

    #循环取得数据，然后作为输入给神经网络进行优化
    for epoch in range(2): # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            '''获取输入；数据是【输入，标签】的列表，将数据放到 NPU 上'''
            inputs, labels = data[0].to(device), data[1].to(device)

            # 将参数梯度归零
            optimizer.zero_grad()

            # 正向+反向+优化
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            prof.step()
            # 打印统计数据
            running_loss += loss.item()
            if i % 2000 == 1999: # 每 2000 小批打印一次
                print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0
```

```
print('Finished Training')
```

步骤 8 代码修改部分包含两部分，开头的部分启用 torch 自带的性能数据收集工具，然后在训练过程中通过 `prof.step()` 收集性能数据，修改部分如下

```
experimental_config = torch_npu.profiler._ExperimentalConfig(
    aic_metrics=torch_npu.profiler.AiCMetrics.PipeUtilization,
    profiler_level=torch_npu.profiler.ProfilerLevel.Level1,
    l2_cache=False
)
with torch_npu.profiler.profile(
    activities=[
        torch_npu.profiler.ProfilerActivity.CPU,
        torch_npu.profiler.ProfilerActivity.NPU
    ],
    record_shapes=True,
    profile_memory=True,
    with_stack=True,
    experimental_config=experimental_config,
    schedule=torch_npu.profiler.schedule(wait=10, warmup=0, active=1, repeat=1),
    on_trace_ready=torch_npu.profiler.tensorboard_trace_handler("./profiling_data")
) as prof:

    # 循环取得数据，然后作为输入给神经网络进行优化
    for epoch in range(2): # Loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            '''获取输入：数据是【输入，标签】的列表，将数据放到NPU上'''
            inputs, labels = data[0].to(device), data[1].to(device)

            # 将参数梯度归零
            optimizer.zero_grad()

            # 正向+反向+优化
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            prof.step()
            # 打印统计数据
```

步骤 9 重启 Kernel，然后执行修改后的代码进行训练，执行完成后可在左侧看到收集的性能数据

 profiling_data

4.2.1.2 性能数据分析

步骤 1 新建一个 Notebook，内核选择 PyTorch-2.1.0，用于执行融合算子分析操作

步骤 2 输入以下代码导入依赖库

```
import sys
sys.path.append("../..")
from prettytable import PrettyTable, ALL
from textwrap import fill
from msprof_analyze.advisor.interface.interface import Interface
```

步骤 3 输入以下代码加载性能数据，根据实际情况替换路径

```
profiling_path = "./profiling_data/xxx /"#根据实际情况替换路径
interface = Interface(profiling_path=profiling_path)
```

步骤 4 指定 profiling 路径后，可以自动识别其中包含的融合算子并给出对应的 torch_npu api 和需要修改的代码堆栈。基于给定堆栈可以快速定位到需要修改的代码段，替换 torch_npu api 后，能够减少 pytorch 侧的小算子的下发，进而提升模型训练速度，输入以下代码进行问题识别并展示问题

```
timeline_fusion_ops_result = interface.get_result("schedule", "timeline_fusion_ops")
display_column_num = 3
problems = timeline_fusion_ops_result.get("问题综述")
problem_table = PrettyTable(problems.get("headers")[:display_column_num])
for row in problems.get("data"):
    for i in range(len(row)):
        row[i] = fill(str(row[i]), width=80)
    problem_table.add_row(row[:display_column_num])

display(problem_table)
```

category	description	suggestion
亲和API接口	目前运行环境版本为cann-8.0.0和torch-2.1.0，发现有1个api接口可以替换。	1. 请根据子表'Affinity training api'替换训练api接口

步骤 5 输入以下代码展示亲和 API 信息和代码堆栈信息

```
fusion_ops_api = timeline_fusion_ops_result.get("亲和 API 接口")
if fusion_ops_api:
    fusion_ops_api_table = PrettyTable(fusion_ops_api.get("headers"))

    for row in fusion_ops_api.get("data"):
        for i in range(len(row)):
            row[i] = fill(str(row[i]), width=80)
        fusion_ops_api_table.add_row(row)

    fusion_ops_api_table.hrules = ALL
    display(fusion_ops_api_table)
```

Affinity API	Code stacks	Stack called counts
torch_npu.optim.NpuFusedSGD	<pre> /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/torch/_ops.py(692): __call__ /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/torch/autograd/profiler.py(631): __enter__ /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/torch/optim/optimizer.py(361): wrapper; /tmp/ipykernel_3656842/1763717862.py(34): <module>; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/interactiveshell.py(3579): run_code; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/interactiveshell.py(3519): run_ast_nodes; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/interactiveshell.py(3336): run_cell_async; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/async_helpers.py(128): _pseudo_sync_runner; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/interactiveshell.py(3132): _run_cell; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/IPython/core/interactiveshell.py(3077): run_cell; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/zmqshell.py(532): run_cell; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/ipkernel.py(353): do_execute; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/kernelbase.py(652): execute_request; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/kernelbase.py(357): dispatch_shell; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/kernelbase.py(450): process_one; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/ipykernel/kernelbase.py(461): dispatch_queue; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/asyncio/events.py(80): _run; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/asyncio/base_events.py(1896): _run_once; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/asyncio/base_events.py(600): run_forever; /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- packages/tornado/platform/asyncio.py(205): start; /home/ma- user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/site- </pre>	1

4.3 思考题

根据分析工具的建议调整代码，然后比对模型训练性能。

答案：根据上面的建议，可替换优化器，重新比对训练时间。

华为认证 AI 系列教程

HCIP-AI-Model Developer

算子开发

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

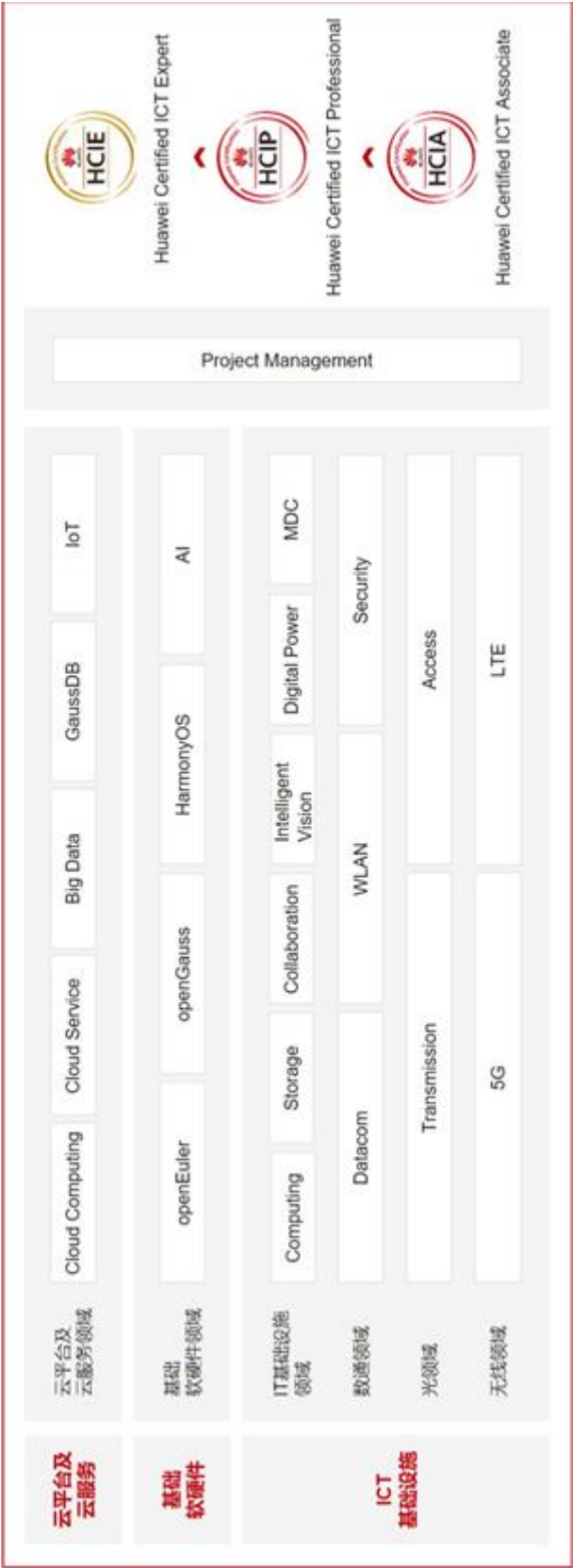
华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

本实验指导书共包含 1 个实验，从初始化 ModelArts 实验环境开发，依次介绍了 Kernel 直调工程执行、标准算子工程创建、AscendCL 单算子调用等内容。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基础的 Linux 与 C++ 知识，同时熟悉昇腾 AI 芯片架构，了解 Ascend C 编程语言；
- 具备一定的数学基础，了解算子计算公式。

实验环境说明

设备介绍

本实验需要使用华为昇腾芯片，并使用配套的 CANN 软件包，为了满足 HCIA-AI-Model Developer 实验需要，建议每套实验环境采用以下配置，以华为云 ModelArts 服务为例：

型号与版本的对应关系如下：

环境名称	实例规格	镜像名称
ModelArts	Ascend: 1*ascend-snt9b1 ARM: 24核 192GB	pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b

准备实验环境

步骤 1 点击以下链接进入 ModelArts 服务，然后点击控制台进入到后台管理界面

<https://www.huaweicloud.com/product/modelarts.html>



步骤 2 点击左上角切换到贵阳一区域



步骤 3 点击左侧开发空间 -> Notebook，然后点击右上角创建 Notebook



步骤 4 名称自定义，然后镜像选择“pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b”，实例规格选择“Ascend: 1*ascend-snt9b1|ARM: 24 核 192GB”，磁盘规格改为 100GB

< | 创建Notebook

☐ mindspore_2.4.10-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-... Ascend Snt9B+ARM算法开发和训练基础镜像，AI引擎预置MindSpore

☒ pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b Ascend_snt9b+ARM算法开发和训练基础镜像，AI引擎预置PyTorch...

* 资源类型 **公共资源池** 专属资源池

* 类型 **ASCEND**

* 实例规格 Ascend: 1*ascend-snt9b1|ARM: 24核 192GB
昇腾规格，适合深度学习场景下的模型训练和调测

* 存储配置 **云硬盘EVS** ②
云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留，存储支持在线按需扩容

* 磁盘规格 - 100 + GB
磁盘规格默认为5GB，从Notebook实例创建成功起，直至删除成功，每GB按照规定费用收费。

步骤 5 创建成功后点击打开即可进入 Notebook 界面

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
准备实验环境	4
2 实验环境介绍	7
2.1 实验介绍	7
2.1.1 关于本实验	7
2.1.2 实验设备介绍	7
3 算子开发	8
3.1 实验介绍	8
3.1.1 关于本实验	8
3.1.2 实验目的	8
3.1.3 实验规划	8
3.2 实验任务配置	8
3.2.1 配置步骤	8
3.3 思考题	14

2 实验环境介绍

2.1 实验介绍

2.1.1 关于本实验

本实验基于华为云 ModelArts 进行，如果使用其他平台或物理服务器，只要软件规格和设备型号满足要求，也可以进行本实验，实验分为两大部分，Kernel 直调工程执行、标准算子工程执行。

2.1.2 实验设备介绍

设备型号与软件版本的对应关系如下：

设备型号	软件名称	软件版本
Atlas A2系列设备 Atlas800 3000	CANN	8.0.0及以上

3 算子开发

3.1 实验介绍

3.1.1 关于本实验

本实验包含两大部分，分别是 Kernel 直调工程执行和标准算子工程执行，其中标准算子工程执行又包括创建标准算子工程和算子编译、算子安装、单算子调用等。

3.1.2 实验目的

- 掌握 Kernel 直调工程的执行流程
- 掌握标准算子工程的创建方法
- 掌握算子编译的方法
- 掌握算子安装的方法
- 掌握单算子调用的流程

3.1.3 实验规划

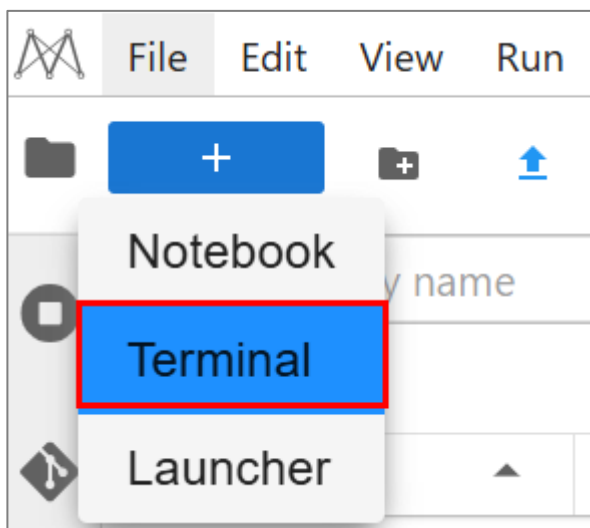
配置 ModelArts 环境 -> 下载示例代码 -> 查看 Kernel 直调工程源码 -> Kernel 直调工程编译验证 -> 创建标准算子工程 -> 查看标准算子工程源码 -> 编译算子 安装算子 单算子调用执行。

3.2 实验任务配置

3.2.1 配置步骤

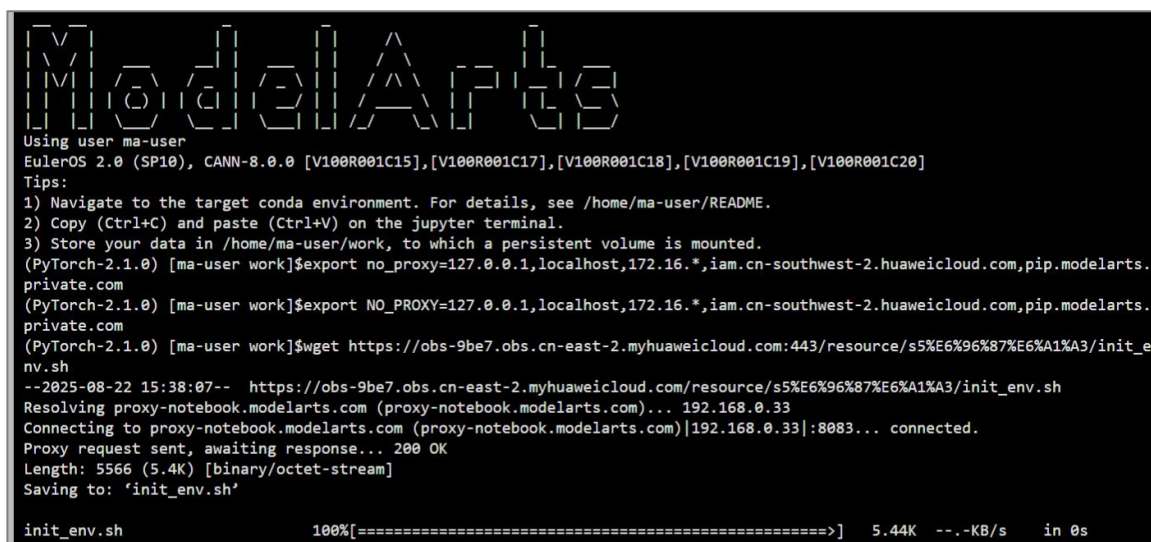
3.2.1.1 配置 ModelArts 环境

步骤 1 进入 Notebook 界面后，点击左上角，新建一个 Terminal 窗口



步骤 2 输入以下命令下载环境初始化脚本

```
export no_proxy=127.0.0.1,localhost,172.16.*,iam.cn-southwest-2.huaweicloud.com,pip.modelarts.private.com
export NO_PROXY=127.0.0.1,localhost,172.16.*,iam.cn-southwest-2.huaweicloud.com,pip.modelarts.private.com
wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com:443/resource/s5%E6%96%87%E6%A1%A3/init_env.sh
#备用下载链接
wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI-Model%20Developer/V2.0/init_env.sh
```



```
ModelArts
Using user ma-user
EulerOS 2.0 (SP10), CANN-8.0.0 [V100R001C15],[V100R001C17],[V100R001C18],[V100R001C19],[V100R001C20]
Tips:
1) Navigate to the target conda environment. For details, see /home/ma-user/README.
2) Copy (Ctrl+C) and paste (Ctrl+V) on the jupyter terminal.
3) Store your data in /home/ma-user/work, to which a persistent volume is mounted.
(PyTorch-2.1.0) [ma-user work]$export no_proxy=127.0.0.1,localhost,172.16.*,iam.cn-southwest-2.huaweicloud.com,pip.modelarts.private.com
(PyTorch-2.1.0) [ma-user work]$export NO_PROXY=127.0.0.1,localhost,172.16.*,iam.cn-southwest-2.huaweicloud.com,pip.modelarts.private.com
(PyTorch-2.1.0) [ma-user work]$wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com:443/resource/s5%E6%96%87%E6%A1%A3/init_env.sh
--2025-08-22 15:38:07-- https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/resource/s5%E6%96%87%E6%A1%A3/init_env.sh
Resolving proxy-notebook.modelarts.com (proxy-notebook.modelarts.com)... 192.168.0.33
Connecting to proxy-notebook.modelarts.com (proxy-notebook.modelarts.com)|192.168.0.33|:8083... connected.
Proxy request sent, awaiting response... 200 OK
Length: 5566 (5.4K) [binary/octet-stream]
Saving to: 'init_env.sh'

init_env.sh      100%[=====] 5.44K --.-KB/s  in 0s
```

步骤 3 输入以下命令执行初始化脚本，该脚本会在当前用户家目录安装 CANN8.1.RC1.beta1 版本，大概执行 8 分钟

```
bash init_env.sh
```

```
gcc/share/locale/sr/LC_MESSAGES/
gcc/share/locale/sr/LC_MESSAGES/gcc.mo
gcc/share/locale/sr/LC_MESSAGES/cpp11.mo
gcc/share/info/
gcc/share/info/libitm.info
gcc/share/info/dir
gcc/share/info/libgomp.info
gcc/share/info/libquadmath.info
gcc/share/man/
gcc/share/man/man7/
gcc/share/man/man7/fsf-funding.7
gcc/share/man/man7/gpl.7
gcc/share/man/man7/gfdl.7
gcc/share/man/man1/
gcc/share/man/man1/gcov.1
gcc/share/man/man1/g++.1
gcc/share/man/man1/gcov-dump.1
gcc/share/man/man1/cpp.1
gcc/share/man/man1/lto-dump.1
gcc/share/man/man1/gcov-tool.1
gcc/share/man/man1/gcc.1
gcc/share/man/man1/gfortran.1

Modellus
Using user ma-user
EulerOS 2.0 (SP10), CANN-8.0.0 [V100R001C15],[V100R001C17],[V100R001C18],[V100R001C19],[V100R001C20]
Tips:
1) Navigate to the target conda environment. For details, see /home/ma-user/README.
2) Copy (Ctrl+C) and paste (Ctrl+V) on the jupyter terminal.
3) Store your data in /home/ma-user/work, to which a persistent volume is mounted.
(PyTorch-2.1.0) [ma-user work]$
```

步骤 4 等待脚本执行成功后，输入以下命令设置算子开发所需环境变量

```
source ~/.bashrc
```

3.2.1.2 下载示例代码（二选一）

可通过下面两种方式下载示例代码，由于 Gitee 仓库目录结构可能会调整，建议优先选择方式一

方式一 通过 OBS 下载

输入以下代码下载并解压示例代码

```
wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIPI-AI-Model%20Developer/V2.0/samples.zip
unzip samples.zip
```

方式二 通过 Gitee 下载

输入以下代码下载并解压示例代码

```
git clone https://gitee.com/ascend/samples.git
```

3.2.1.3 Kernel 直调工程执行

步骤 1 输入以下命令进入 Kernel 直调工程示例目录

```
cd samples/operator/ascendc/0_introduction/3_add_kernellaunch/AddKernelInvocationNeo
```

```
(PyTorch-2.1.0) [ma-user work]$cd samples/operator/ascendc/0_introduction/3_add_kernellaunch/AddKernelInvocationNeo
(PyTorch-2.1.0) [ma-user AddKernelInvocationNeo]$
```

步骤 2 可使用 cat 命令查看目录下源码文件

步骤 3 输入以下命令查看 NPU 信息

```
npu-smi info
```

```
(PyTorch-2.1.0) [ma-user AddKernelInvocationNeo]$npu-smi info
+-----+
| npu-smi 23.0.6 | Version: 23.0.6 |
+-----+
| NPU Name | Health | Power(W) | Temp(C) | Hugepages-Usage(page) |
| Chip | Bus-Id | AICore(%) | Memory-Usage(MB) | HBM-Usage(MB) |
+-----+
| 1 | | OK | 88.8 | 41 | 0 / 0 |
| 0 | 0000:01:00.0 | 0 | 0 / 0 | 2847 / 32768 |
+-----+
+-----+
| NPU Chip | Process id | Process name | Process memory(MB) |
+-----+
| No running processes found in NPU 1 |
+-----+
```

步骤 4 输入以下命令进行编译验证（xxx 部分替换为上一步显示的型号）

```
bash run.sh -r npu -v Ascendxxx
```

步骤 5 查看输出

```
-- Installing: /home/ma-user/work/samples/operator/ascendc/0_introduction/3_add_kernellaunch,
de/ascendc_kernels_npu
-- Installing: /home/ma-user/work/samples/operator/ascendc/0_introduction/3_add_kernellaunch,
de/ascendc_kernels_npu/aclrtlaunch_add_custom.h
-- Installing: /home/ma-user/work/samples/operator/ascendc/0_introduction/3_add_kernellaunch,
de/ascendc_kernels_npu/aclrtlaunch_triple_chevrons_func.h
-- Installing: /home/ma-user/work/samples/operator/ascendc/0_introduction/3_add_kernellaunch,
scendc_kernels_bbit
9c88ab553b9f626d71878661682bbe87 output/golden.bin
9c88ab553b9f626d71878661682bbe87 output/output_z.bin
error ratio: 0.0000, tolerance: 0.0010
test pass
(PyTorch-2.1.0) [ma-user AddKernelInvocationNeo]$
```

3.2.1.4 标准算子工程执行

步骤 1 输入以下命令进入到标准算子工程目录

```
cd ~/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch
```

步骤 2 输入以下命令查看 AddCustom.json 文件内容，下一步会根据这个文件生成标准算子工程

```
cat AddCustom.json
```

```
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$cat AddCustom.json
[
  {
    "op": "AddCustom",
    "input_desc": [
      {
        "name": "x",
        "param_type": "required",
        "format": [
          "ND"
        ],
        "type": [
          "float16"
        ]
      },
      {
        "name": "y",
        "param_type": "required",
        "format": [
          "ND"
        ],
        "type": [
          "float16"
        ]
      }
    ],
    "output_desc": [
      {
        "name": "z",
        "param_type": "required",
        "format": [
          "ND"
        ],
        "type": [
          "float16"
        ]
      }
    ]
  }
]
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$
```

步骤 3 输入以下命令在当前用户家目录生成标准算子工程（xxx 替换为实际型号）

```
`${ASCEND_TOOLKIT_HOME}/python/site-packages/bin/msopgen gen -i AddCustom.json -c ai_core-Ascendxxx -lan cpp -out ~/Addcustom
```

```
2025-08-22 16:09:15 (172815) - [INFO] Start to generate AI Core operator files.
2025-08-22 16:09:15 (172815) - [INFO] Start to parse the ir template:/home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/AddCustom.json
2025-08-22 16:09:15 (172815) - [INFO] Start to parse the op: AddCustom
2025-08-22 16:09:15 (172815) - [INFO] Start to parse the input_desc: x
2025-08-22 16:09:15 (172815) - [INFO] Start to parse the input_desc: y
2025-08-22 16:09:15 (172815) - [INFO] Start to parse the output_desc: z
2025-08-22 16:09:15 (172815) - [WARNING] The "attr" value is invalid or no "attr" exists in the map.
2025-08-22 16:09:15 (172815) - [INFO] Start to check the type and format between the inputs/outputs in IR template.
2025-08-22 16:09:15 (172815) - [INFO] Start to generate a new project.
2025-08-22 16:09:15 (172815) - [WARNING] File /home/ma-user/Addcustom/cmake/config.cmake already exists and will be overwrite!
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/cmake/config.cmake generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/op_host/add_custom_tiling.h generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/op_host/add_custom.cpp generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/op_kernel/add_custom.cpp generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/framework/tf_plugin/tensorflow_add_custom_plugin.cc generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] File /home/ma-user/Addcustom/framework/tf_plugin/CMakeLists.txt generated successfully.
2025-08-22 16:09:15 (172815) - [INFO] Generation completed.
```

步骤 4 输入以下命令进入到上一步生成的标准算子工程目录

```
cd ~/Addcustom
```

```
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$cd ~/Addcustom
(PyTorch-2.1.0) [ma-user Addcustom]$
```

步骤 5 可使用 cat 命令查看当前目录下源码文件

步骤 6 输入以下命令回到已有的标准算子工程目录

```
cd ~/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch
```

步骤 7 可以查看当前目录下 AddCustom 下的源码文件

步骤 8 输入以下命令编译算子（xxx 替换为实际型号）

```
bash install.sh -v Ascendxxx
```

```
[ 91%] Built target cust_tf_parsers
[100%] Linking CXX shared library libcust_opsproto_rt2.0.so
[100%] Built target cust_op_proto
Run CPack packaging tool...
CPack: Create package using External
CPack: Install projects
CPack: - Run preinstall target for: opp
CPack: - Install project: opp []
CPack: Create package

About to compress 724 KB of data...
Adding files to archive named "custom_opp_euleros_aarch64.run"...
./help.info
./install.sh
./packages/vendors/customize/framework/tensorflow/libcust_tf_parsers.so
./packages/vendors/customize/framework/tensorflow/npv_supported_ops.json
./packages/vendors/customize/op_api/include/acInn_add_custom.h
./packages/vendors/customize/op_api/lib/libcust_opapi.so
./packages/vendors/customize/op_impl/ai_core/tbe/config/ascend910b/aic-ascend910b-ops-info.json
./packages/vendors/customize/op_impl/ai_core/tbe/customize_impl/dynamic/add_custom.cpp
./packages/vendors/customize/op_impl/ai_core/tbe/customize_impl/dynamic/add_custom.py
./packages/vendors/customize/op_impl/ai_core/tbe/kernel/ascend910b/add_custom/AddCustom_ab1b6750d7f510985325b603cb06dc8b.o
./packages/vendors/customize/op_impl/ai_core/tbe/kernel/ascend910b/add_custom/AddCustom_ab1b6750d7f510985325b603cb06dc8b.o
./packages/vendors/customize/op_impl/ai_core/tbe/kernel/config/ascend910b/add_custom.json
./packages/vendors/customize/op_impl/ai_core/tbe/kernel/config/ascend910b/binary_info_config.json
./packages/vendors/customize/op_impl/ai_core/tbe/op_tiling/lib/linux/aarch64/libcust_opmaster_rt2.0.so
./packages/vendors/customize/op_impl/ai_core/tbe/op_tiling/lib/liboptiling.so
./packages/vendors/customize/op_proto/inc/op_proto.h
./packages/vendors/customize/op_proto/lib/linux/aarch64/libcust_opsproto_rt2.0.so
./packages/vendors/customize/version.info
./upgrade.sh
CRC: 3587977614
SHA256: 64620b8c80ac35c58e0089591185f0e6438200fe8afbeddb1527849c3f134d
Skipping md5sum at user request

Self-extractable archive "custom_opp_euleros_aarch64.run" successfully created.
Copy /home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/CustomOp/build_out/_CPack_Packages/Linux/External/custom_opp_euleros_aarch64.run to /home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/CustomOp/build_out/
CPack: - package: /home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/CustomOp/build_out/custom_opp_euleros_aarch64.run.json generated
CPack: - package: /home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/CustomOp/build_out/custom_opp_euleros_aarch64.run generated.
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$
```

步骤 9 输入以下命令查看生成的算子安装包

```
ls CustomOp/build_out/
```

```
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$ls CustomOp/build_out/
autogen          cmake_install.cmake  CPackSourceConfig.cmake  framework  op_host  version.info
CMakeCache.txt   CPackConfig.cmake    custom_opp_euleros_aarch64.run  install_manifest.txt  op_kernel
CMakeFiles       _CPack_Packages      custom_opp_euleros_aarch64.run.json  Makefile    scripts
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$
```

步骤 10 输入以下命令安装算子（以实际算子包名称为准）

```
./CustomOp/build_out/custom_opp_euleros_aarch64.run
```

```
(PyTorch-2.1.0) [ma-user 1_add_frameworklaunch]$./CustomOp/build_out/custom_opp_euleros_aarch64.run
Verifying archive integrity... 100% SHA256 checksums are OK. All good.
Uncompressing version:1.0 100%
[ops_custom] [2025-08-22 16:16:40] [WARNING] The directory /home/ma-user/Ascend/ascend-toolkit/latest/opp does not have sufficient permissions. Please check and modify the folder permissions (e.g., using chmod), or use the --install-path option to specify an installation path and change the environment variable ASCEND_CUSTOM_OPP_PATH to the specified path.
[ops_custom] [2025-08-22 16:16:40] [INFO] copy uninstall sh success
[ops_custom] [2025-08-22 16:16:40] [INFO] upgrade framework
[ops_custom] [2025-08-22 16:16:40] [INFO] create /home/ma-user/Ascend/ascend-toolkit/latest/opp/vendors/customize/framework.
[ops_custom] [2025-08-22 16:16:40] [INFO] copy new ops framework files .....
[ops_custom] [2025-08-22 16:16:40] [INFO] upgrade op proto
[ops_custom] [2025-08-22 16:16:40] [INFO] create /home/ma-user/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_proto.
[ops_custom] [2025-08-22 16:16:41] [INFO] copy new ops op proto files .....
[ops_custom] [2025-08-22 16:16:41] [INFO] upgrade op impl
[ops_custom] [2025-08-22 16:16:41] [INFO] create /home/ma-user/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_impl.
[ops_custom] [2025-08-22 16:16:41] [INFO] copy new ops op_impl files .....
[ops_custom] [2025-08-22 16:16:41] [INFO] upgrade op api
[ops_custom] [2025-08-22 16:16:41] [INFO] create /home/ma-user/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_api.
[ops_custom] [2025-08-22 16:16:41] [INFO] copy new ops op_api files .....
[ops_custom] [2025-08-22 16:16:41] [INFO] upgrade version.info
[ops_custom] [2025-08-22 16:16:41] [INFO] copy new version.info files .....
[ops_custom] [2025-08-22 16:16:41] [INFO] no need to upgrade custom.proto files
[ops_custom] [2025-08-22 16:16:41] [INFO] using requirements: when custom module install finished or before you run the custom module, execute the command [ export LD_LIBRARY_PATH=/home/ma-user/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_api/lib/:${LD_LIBRARY_PATH} ] to set the environment path
SUCCESS
```

步骤 11 输入以下命令使用 AscendCL 调用执行算子

```
cd AclNNInvocation && bash run.sh
```

```
INFO: cmake success!
[ 20%] Building CXX object CMakeFiles/execute_add_op.dir/operator_desc.cpp.o
[ 40%] Building CXX object CMakeFiles/execute_add_op.dir/op_runner.cpp.o
[ 60%] Building CXX object CMakeFiles/execute_add_op.dir/main.cpp.o
[ 80%] Building CXX object CMakeFiles/execute_add_op.dir/common.cpp.o
[100%] Linking CXX executable /home/ma-user/work/samples/operator/ascendc/0_introduction/1_add_frameworklaunch/AclNNInvocation/output/execute_add_op
[100%] Built target execute_add_op
INFO: make success!
INFO: execute op!
[INFO] Set device[0] success
[INFO] Get RunMode[1] success
[INFO] Init resource success
[INFO] Set input success
[INFO] Copy input[0] success
[INFO] Copy input[1] success
[INFO] Create stream success
[INFO] Execute aclnnAddCustomGetWorkspaceSize success, workspace size 0
[INFO] Execute aclnnAddCustom success
[INFO] Synchronize stream success
[INFO] Copy output[0] success
[INFO] Write output success
[INFO] Run op success
[INFO] Reset Device success
[INFO] Destroy resource success
INFO: acl executable run success!
error ratio: 0.0000, tolerance: 0.0010
test pass
```

3.3 思考题

msopgen 工具的作用是什么？

答案：根据 JSON 文件创建标准算子工程。

华为认证 AI 系列教程

HCIP-AI-Model Developer

集合通信测试

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

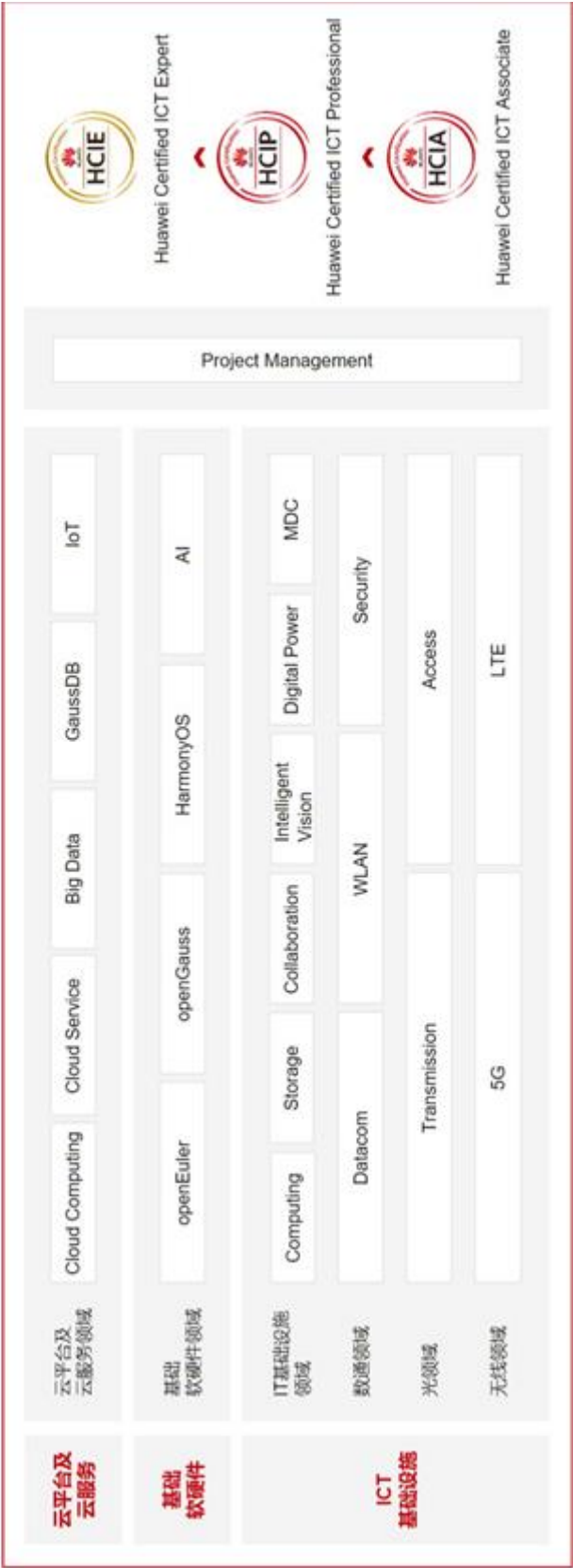
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

本实验指导书共包含 1 个实验，集合通信测试。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本 Linux 知识，了解 AI 芯片和设备的基础知识。

实验环境说明

设备介绍

本实验需要使用华为昇腾芯片，并使用配套的 CANN 软件包，同时为进行卡间测试，至少需要两块儿物理 NPU，为了满足 HCIA-AI-Model Developer 实验需要，建议每套实验环境采用以下配置，以华为云 ModelArts 服务为例：

型号与版本的对应关系如下：

环境名称	实例规格	镜像名称
ModelArts	Ascend: 2*ascend-snt9b1 ARM: 24核 192GB	pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b

准备实验环境

步骤 1 点击以下链接进入 ModelArts 服务，然后点击控制台进入到后台管理界面

<https://www.huaweicloud.com/product/modelarts.html>



步骤 2 点击左上角切换到贵阳一区域



步骤 3 点击左侧开发空间 -> Notebook，然后点击右上角创建 Notebook



步骤 4 名称自定义，然后镜像选择“pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b”，实例规格选择“Ascend: 2*ascend-snt9b1|ARM: 24 核 192GB”，磁盘规格默认即可

< | 创建Notebook

	☐	mindspore_2.4.10-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-...	Ascend Snt9B+ARM算法开发和训练基础镜像, AI引擎预置MindSpore
	☒	pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend_snt9b+ARM算法开发和训练基础镜像, AI引擎预置PyTorch...
* 资源类型	公共资源池 专属资源池		
* 类型	ASCEND		
* 实例规格	Ascend: 2*ascend-snt9b1 ARM: 48核 384GB 昇腾规格, 适合深度学习场景下的模型训练和调测		
* 存储配置	云硬盘EVS ? 云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下, 该目录下的内容在实例停止后会被保留, 存储支持在线按需扩容 * 磁盘规格 - 5 + GB 磁盘规格默认为5GB, 从Notebook实例创建成功起, 直至删除成功, 每GB按照规定费用收费。		

步骤 5 创建成功后点击打开即可进入 Notebook 界面。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
准备实验环境	3
2 实验环境介绍	7
2.1 实验介绍	7
2.1.1 关于本实验	7
2.1.2 实验设备介绍	7
3 集合通信测试	8
3.1 实验介绍	8
3.1.1 关于本实验	8
3.1.2 实验目的	8
3.1.3 实验规划	8
3.2 实验任务配置	8
3.2.1 配置步骤	8
3.3 思考题	11

2 实验环境介绍

2.1 实验介绍

2.1.1 关于本实验

本实验基于华为云 ModelArts 进行，如果使用其他平台或物理服务器，只要软件规格和设备型号满足要求，也可以进行本实验，由于集合通信测试需要集群，本实验进行了简化，测试机内卡与卡、卡和主机等之间的带宽。

2.1.2 实验设备介绍

设备型号与软件版本的对应关系如下：

设备型号	软件名称	软件版本
Atlas A2 系列设备 (要求双卡)	CANN	8.0.0 及以上

3 集合通信测试

3.1 实验介绍

3.1.1 关于本实验

本实验共包含两部分，MindCluster 组件的安装，通信带宽测试。

3.1.2 实验目的

- 掌握如何利用 ascend-dmi 工具进行通信带宽测试

3.1.3 实验规划

安装 MindCluster 组件 -> 进行通信带宽测试。

3.2 实验任务配置

3.2.1 配置步骤

步骤 1 点击以下链接进入昇腾官网，下载 MindCluster 组件安装包

<https://www.hiascend.com/developer/download/community/result?module=dl+cann&cluster=6.0.0&cann=8.0.0.beta1>

配套资源

版本选择 ①

MindCluster 6.0.0

CANN 8.0.0.beta1

CPU架构

☐ 全部 ☒ AArch64 ☐ X86_64

← 返回资源选择

资源下载遇到问题，您可以查看 [常见问题](#) 或 [提交工单](#)

昇腾社区提供的软件受其对应协议的约束，您可通

查找结果

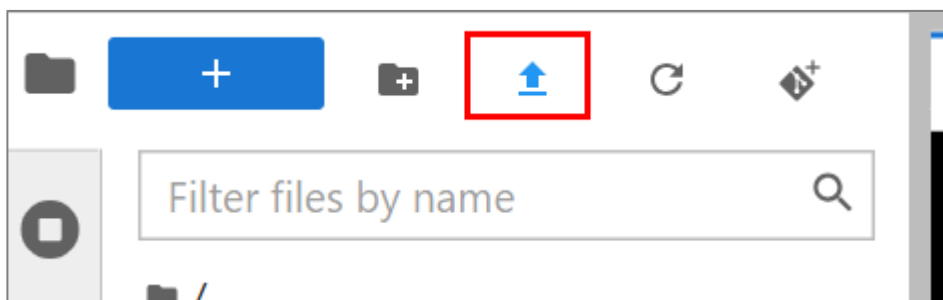
MindCluster 2024/12/31

[查看文档](#) | [软件包](#) | [版本信息](#)

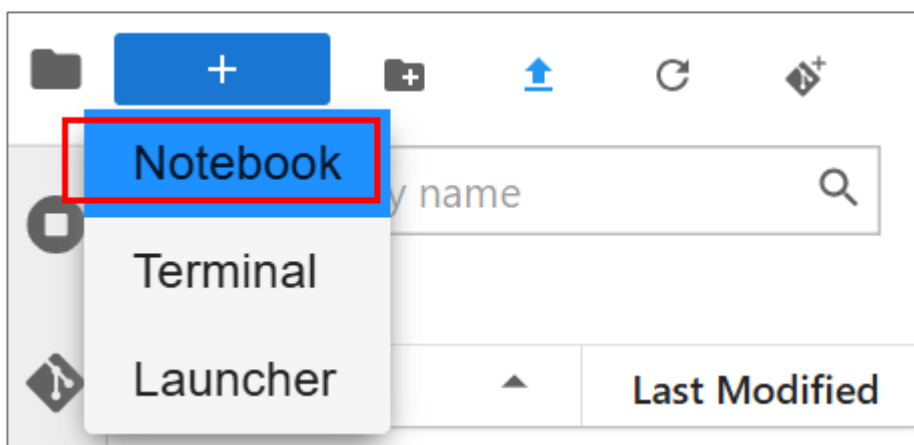
☐ 全部 ☒ ToolBox ☐ 集群调度 ☐ 故障诊断

软件包名称	说明	操作
☐ Ascend-mindx-toolbox_6.0.0.SPC1_linux-aarch64.deb	Ascend DMI性能测试工具、日志收集工具等	下载软件包 下载数字签名
☐ Ascend-mindx-toolbox_6.0.0.SPC1_linux-aarch64.run	Ascend DMI性能测试工具、日志收集工具等	下载软件包 下载数字签名
☐ Ascend-mindx-toolbox_6.0.0.SPC1_linux-aarch64.zip	Ascend DMI性能测试工具、日志收集工具等	下载软件包 下载数字签名

步骤 2 下载完成后返回 Notebook 界面，点击左上角文档上传按钮，上传软件包



步骤 3 上传完成后点击左上角创建 Terminal



步骤 4 输入以下命令安装组件，安装过程中需要输入 Y 确认

```
chmod a+x Ascend-mindx-toolbox_6.0.0.SPC1_linux-aarch64.run
./Ascend-mindx-toolbox_6.0.0.SPC1_linux-aarch64.run --install
```

步骤 5 安装完成后输入以下命令进行测试

```
#该命令默认使用 Device 0，以 h2d、d2h、d2d 三个数据流向和步长模式显示的带宽耗时信息
ascend-dmi --bw -q
```

```
Host to Device Test
Device 0: /dev/xrt/xrt0

-----
Size(Bytes)  Execute Times  Bandwidth(GB/s)  Elapsed Time(us)
-----
2            5
4            5
8            5
16           5
32           5
64           5
128          5
256          5
512          5
1024         5
2048         5
4096         5
8192         5
16384        5
32768        5
65536        5
131072       5
262144       5
524288       5
1048576      5
2097152      5
4194304      5
8388608      5
16777216     5
33554432     5
-----
```

```
#该命令指定 Device 0，指定 size 为 128M，定长模式迭代 100 次，测试 h2d 带宽
ascend-dmi --bw -t h2d -d 0 --et 100 -s 134217728 -q

#该命令指定 Device 0，指定 size 为 128M，定长模式迭代 100 次，测试 d2h 带宽
ascend-dmi --bw -t d2h -d 0 --et 100 -s 134217728 -q

#该命令测试数据从 Device 侧传输到同一 Device 侧的带宽与总耗时
ascend-dmi --bw -t d2d -d 0 -q

#该命令测试数据从源头 Device 0 传输到目标 Device 1，执行次数为 100，size 为 128M 的 p2p 测试
ascend-dmi --bw -t p2p --dd 0 --ds 1 --et 100 -s 134217728 -q

#该命令为不指定源头 Device 和目标 Device 的 p2p 测试
ascend-dmi --bw -t p2p -q
```

3.3 思考题

不同的带宽（如 d2d、d2h）对模型执行的哪些流程会产生影响？

答案：d2d 包括卡间互联，对应张量并行训练的效率影响较高，同设备的 d2d 则为内存带宽，对 LLM 推理的 Decode 阶段影响较高，d2h 和 h2d 的数据交互发生在 Device 和 Host 侧，影响数据处理、模型文件保存、算子下发等。

华为认证 AI 系列教程

HCIP-AI-Model Developer

YOLOv5预训练

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

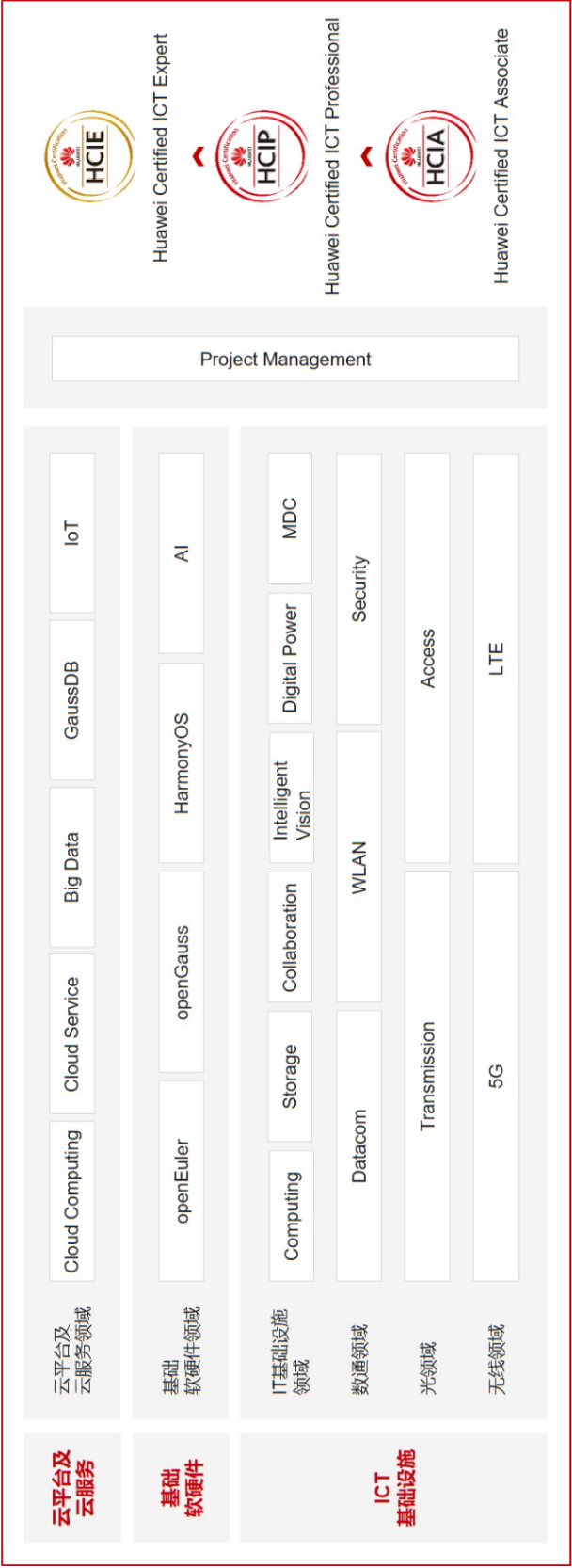
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

- 本实验指导书共包含 1 个实验，YOLOv5 的预训练。

实验环境说明

组网说明

本实验环境用于 HCIP-AI Model Developer V2.0 预训练实验。每套实验环境包括一台 windows 系统电脑，一个华为云 OBS 桶，一台 ModelArts 服务器。每套实验环境适用于 1 位学员操作。

设备介绍

为了满足 HCIP-AI Model Developer V2.0 实验需要，建议每套实验环境采用以下配置：

设备名称、型号与版本的对应关系如下：

设备名称	区域/设备型号
OBS桶	西南-贵阳一
obsutil	obsutil_windows64
ModelArts	Ascend: 1*ascend-snt9b1 ARM: 24核 192GB



目录

前 言	3
简介	3
内容描述	3
实验环境说明	3
1 基于 YOLOv5 的模型预训练	5
1.1 实验介绍	5
1.1.1 关于本实验	5
1.2 实验准备步骤	5
1.2.1 下载 coco2017 数据集	5
1.2.2 上传文件到 OBS	7
1.2.3 创建 ModelArts 实例	11
1.2.4 下载数据和模型代码到服务器	14
1.2.5 安装依赖包	16
1.2.6 处理数据集	17
1.3 训练模型	21
1.3.1 YOLOv5 介绍	21
1.3.2 启动训练	21
1.4 思考题	40

1 基于 YOLOv5 的模型预训练

1.1 实验介绍

1.1.1 关于本实验

YOLOv5 是一种先进且高效的目标检测算法，专为实现快速而准确的物体识别设计。本实验是使用 PyTorch 框架，在昇腾服务器上预训练 YOLOv5 模型。通过此实验，学员将了解 YOLOv5 目标检测的原理，网络结构，数据预处理过程，并掌握模型训练流程以及使用训练模型进行目标检测等。

1.2 实验设备介绍

设备型号与软件版本的对应关系如下，本实验中使用 ModelArts 提供的镜像。如果使用其他昇腾设备，环境相关适配软件请参见下表，昇腾适配软件请参照官网适配版本安装：

硬件设备型号	软件名称	软件版本
Atlas A2 系列训练设备	CANN	8.0.0
	Python	3.10
	PyTorch	2.1.0

1.3 实验准备

1.3.1 下载 coco2017 数据集

预训练实验所需要的 coco2017 数据集，本次下载 Train2017、Val2017 两个数据集和一个标注文件 annotations_trainval2017.zip。

1.3.1.1 下载方式 1（首选）

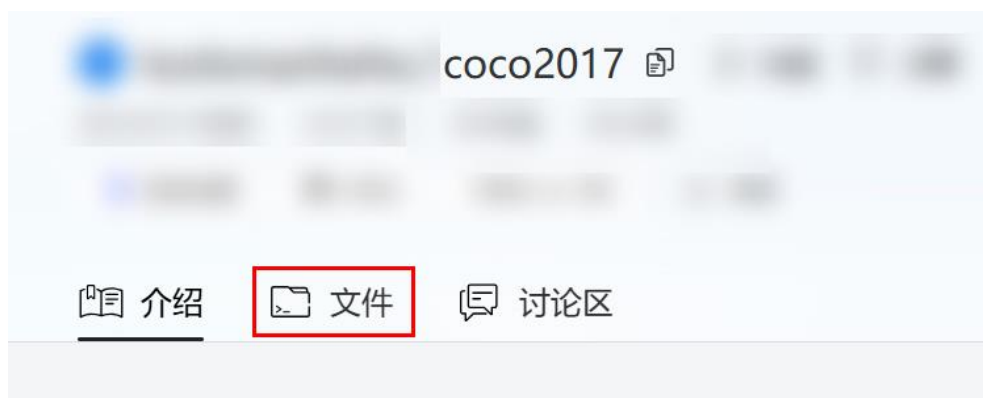
步骤 1 登陆魔乐社区（华为合作网站）：<https://modelers.cn/>。

步骤 2 在网页上方选择“数据集”选项，点击进入。

步骤 3 在“全部数据集”下的搜索框输入“coco2017”，回车，选择第一个数据集，点击进入。



步骤 4 点击“文件”。



步骤 5 找到红框内对应的文件，点击下载文件。

名称	大小	格式	描述	时间	操作
annotations_trainval2017.zip	0.24 GB	LFS	add test val annotations	1个月前	下载文件
train2017.zip	18.01 GB	LFS	Upload folder using openMind...	1个月前	下载文件
val2017.zip	0.76 GB	LFS	add test val annotations	1个月前	下载文件

步骤 6 将下载的文件放在同一个文件夹下，记录存放路径。

1.3.1.2 下载方式二（官方链接）

步骤 1 登陆 COCO 数据集下载官网：<https://cocodataset.org/#download>。

步骤 2 找到所需的三个文件：2017 Train images [118K/18GB]、2017 Val images [5K/1GB]、2017 Train/Val annotations [241MB]，分别点击下载，或者右击选择“复制链接地址”，然后粘贴在浏览器地址栏，下载文件。

步骤 3 归档好下载数据，记住保存路径。

1.3.1.3 下载方式三（备选，需要注册 kaggle 账户）

步骤 1 登陆 kaggle 数据集官网：<https://www.kaggle.com/datasets>。

步骤 2 在“Search datasets”栏输入“coco2017”，选在出现在第一位的 coco2017 数据集，数据集大小约 20G，点击进入数据集。

步骤 3 数据集目录如下，请确认是否选择正确。

```
annotations_trainval2017
train2017
val2017
```

总计大小约 20.96 GB

步骤 4 点击“Download”按钮，选择“Download dataset as zip（20G）”，然后登陆 kaggle 账户，如果没有，可以选择注册账号后登陆。

步骤 5 保存数据，记住保存路径。

1.3.2 上传文件到 OBS

1.3.2.1 创建 OBS 桶

步骤 1 登陆华为云官网：<https://www.huaweicloud.com/>，登陆华为云账号。

步骤 2 选择 OBS。



步骤 3 点击控制台，进入桶列表。



步骤 4 创建桶。

点击创建桶，



区域选择“西南-贵阳一”，



输入桶名称，选择“单 AZ 存储”，

桶配置

桶名称

请输入桶名称

[查看命名规则](#)

✖ 此项不能为空。

桶访问域名: <名称> .obs.cn-southwest-2.myhuaweicloud.com

数据冗余存储策略 ?

多AZ存储

单AZ存储

创建成功后**不支持修改**。

数据在单个可用区 (AZ) 中存储, 如果您对数据可用性有更高要求, 建议选择多AZ。

左选半只

其余选项保持不变, 点击右下方“立即创建”, 创建成功, 自动返回桶列表。



1.3.2.2 下载 obsutil

步骤 1 进入 OBS 说明文档: https://support.huaweicloud.com/utiltg-obs/obs_11_0003.html。

步骤 2 下载 Windows 版和 linux ARM 版。

表1 下载地址

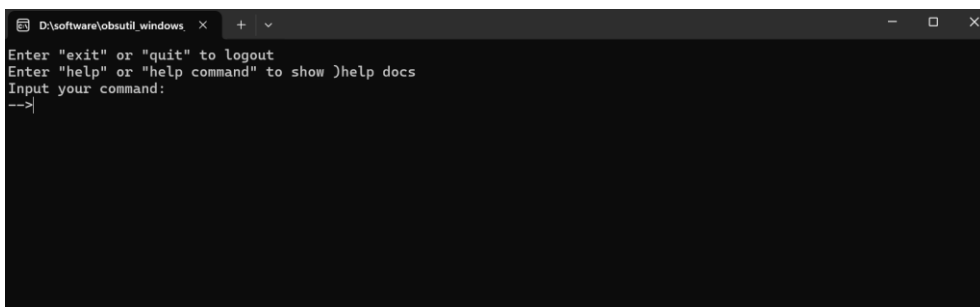


操作系统	软件包下载地址	SHA256校验和
Windows 64位	obsutil_windows64	3cf48e94f669e2b6035a10558d9fd 2d254e530260b9caf0dd811bb1a9 0e1815f
Linux AMD 64位 (Linux x86 64位)	obsutil_linux_amd64	077518c3e3b82eaedbbdacc8e955 1f3a901f7835172c654492b06cd9a 733da35
Linux ARM 64位	obsutil_linux_arm64	68ab4fb65718595ae855b2fb579c8 b6da2b06727477ee75d6513ee224 cd821aa
macOS 64位	obsutil_mac64	1c2785c8641d64a5e00848222c31 8217053220e1641634a96e603f15 72f2f238

Windows 64 位用于 Windows 系统上传或下载数据，Linux ARM 64 位用于 ModelArts 上上传或下载数据。

步骤 3 打开 obsutil

解压 obsutil_windows64，解压后双击.exe 文件，启动 obsutil。



1.3.2.3 上传文件到 OBS

1.3.2.3.1 创建访问密钥

OBS 通过用户账号中的 AK 和 SK 进行签名验证，确保通过授权的账号才能访问指定的 OBS 资源。以下是对 AK 和 SK 的解释说明：

Access Key Id (AK)：访问密钥 ID。与私有访问密钥关联的唯一标识符；访问密钥 ID 和私有访问密钥一起使用，对请求进行加密签名。

Secret Access Key (SK)：与访问密钥 ID 结合使用的私有访问密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

创建访问密钥的操作步骤如下：

步骤 1 登陆华为云，输入账号、密码，在控制台单击页面右上角的用户名，并选择“我的凭证”。

步骤 2 在“我的凭证”页面，单击左侧导航栏的“访问密钥”。

步骤 3 在“访问密钥”页面，单击“新增访问密钥”。

步骤 4 单击“确定”。

步骤 5 在弹出的“下载确认”提示框中，单击“确定”后，密钥会直接保存到浏览器默认的下载文件夹中。

步骤 6 打开下载下来的“credentials.csv”文件既可获取到访问密钥（AK 和 SK）。

1.3.2.3.2 初始化 obsutil

步骤 1 在 1.2.2.2 步骤 3 中的界面输入。

```
config -interactive
```

步骤 2 然后输入已经下载的 AK 和 SK（输入时不显示 SK）。

步骤 3 输入 endpoint。

```
https://obs.cn-southwest-2.myhuaweicloud.com
```

步骤 4 提示输入 token 时，直接按回车跳过，初始化完成。

步骤 5 测试连通性。

```
obsutil ls -s
```

根据命令回显结果，检查配置结果：

如果返回结果中包含“Bucket number :”，表明配置正确。

1.3.2.3.3 上传数据

使用以下命令将下载的文件上传到 OBS。

```
obsutil cp 本地存储路径\文件全称 obs://桶名称
```

例如：

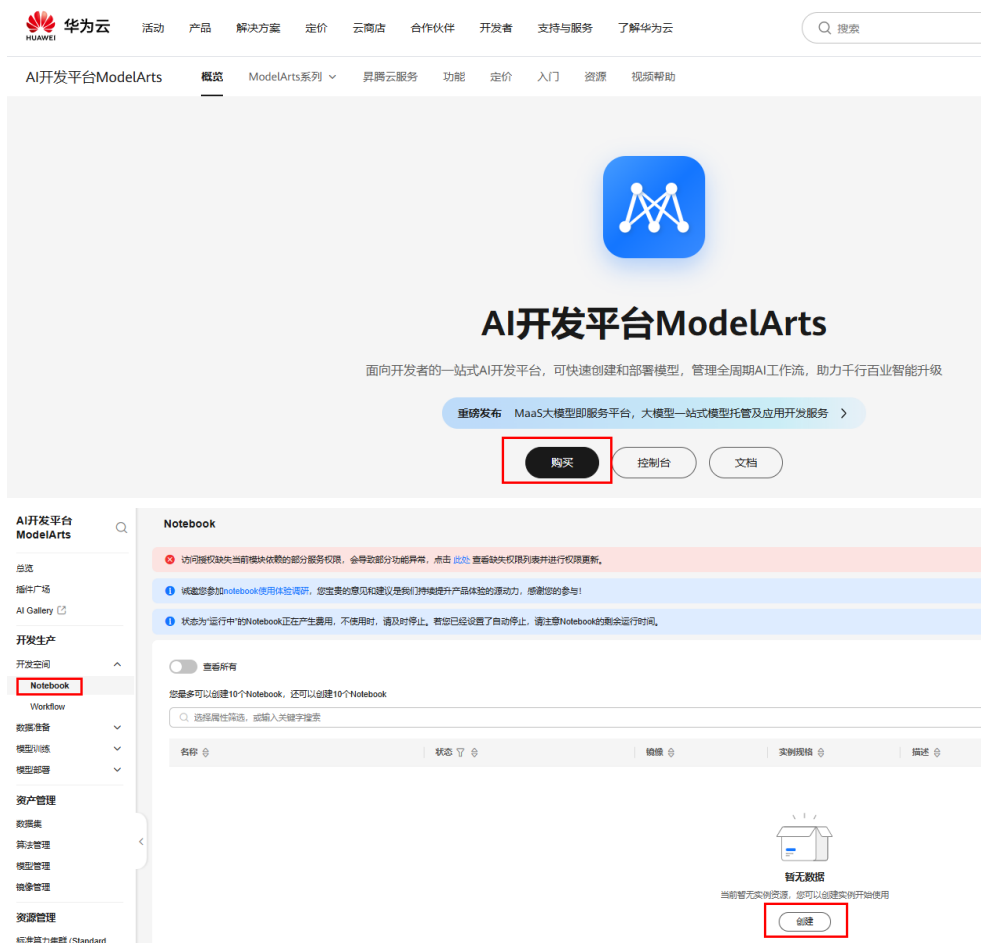
```
obsutil cp D:\Download\annotations_trainval2017 obs://data
```

依次把下载的数据集上传到 OBS 上。

1.3.3 创建 ModelArts 实例

步骤 1 登陆 ModelArts: <https://www.huaweicloud.com/product/modelarts.html>。

步骤 2 购买 ModelArts 上 Notebook，点击购买，进入购买界面，选择 Notebook，点击创建。



步骤 3 选择区域为“西南-贵阳一”，对 Notebook 命名，设置自动启停时间，按需设置时长。



步骤 4 镜像、服务器及硬盘大小按下图选择。

★ 镜像 预置镜像 自定义镜像 前往AI Gallery获取更多镜像

名称	描述
☐ pytorch_2.1.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend_snt9b+ARM算法开发和训练基础镜像, AI引擎预置PyTorch...
☐ mindspore_2.4.0-cann_8.0.rc3-py_3.9-euler_2.10.10-aarch64-snt9b	Ascend Snt9b+ARM算法开发和训练基础镜像, AI引擎预置MindSpore...
☐ mindspore_2.4.10-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend Snt9b+ARM算法开发和训练基础镜像, AI引擎预置MindSpore...
☒ pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend_snt9b+ARM算法开发和训练基础镜像, AI引擎预置PyTorch...
☐ pytorch_2.1.0-uda_12.1-py_3.10.6-ubuntu_22.04-x86_64	cuda12.1+X86算法开发和训练基础镜像, AI引擎预置PyTorch2.1
☐ pytorch_2.1.0-cann_8.1.rc1-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend_snt9b+ARM算法开发和训练基础镜像, AI引擎预置PyTorch...
☐ mindspore_2.6.0rc1-cann_8.1.rc1-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend Snt9b+ARM算法开发和训练基础镜像, AI引擎预置MindSpore...
☐ pytorch_2.1.0-cann_8.0.rc3-py_3.9-euler_2.10.10-aarch64-snt9b	Ascend_snt9b+ARM算法开发和训练基础镜像, AI引擎预置PyTorch...

总条数: 18 10 < 1 2 >

★ 资源类型 公共资源池 专属资源池

★ 类型 ASCEND

★ 实例规格 Ascend: 1*ascend-snt9b3(ARM: 24核 192GB)
昇腾规格, 适合深度学习场景下的模型训练和推理

★ 存储配置 云硬盘EVS 云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下, 该目录下的内容在实例停止后会被保留, 存储支持在线扩容

★ 磁盘规格 - 120 + GB
磁盘规格默认为5GB, 从Notebook实例创建成功起, 直至删除成功, 每GB按照标准费用收费。

步骤 5 点击右下角“立即创建”，然后点击“提交”，点击“立即返回”。

步骤 6 等待创建成功后，点击打开，进入服务器实例。

查看所有

您最多可以创建10个Notebook, 还可以创建9个Notebook

选择属性筛选, 或输入关键字搜索

名称	状态	镜像	实例规格	描述	创建时间	创建者	操作
pytorch_2.1.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	运行中 (58分钟后自动停止)	Ascend: 1*ascend-snt9b3(ARM: 24核 192GB)	-	-	-	-	打开 回退 停止 更多

点击下图中的 Terminal。

File Edit View Run Kernel Git Tabs Settings Help

Filter files by name

Name Last Modified

- .modelarts 4 minutes ago
- lost-found 4 minutes ago

Launcher

Notebook

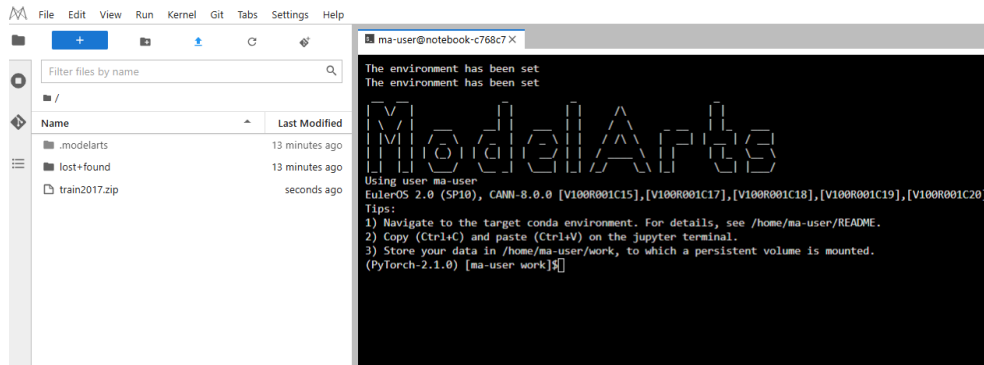
- PyTorch-2.1.0
- python-3.10.6

Console

- PyTorch-2.1.0
- python-3.10.6

Other

- Terminal
- VS Code
- Performance Advisor
- Text File
- Markdown File
- Pipeline As Code
- TensorBoard
- Python File
- Notebook Jobs



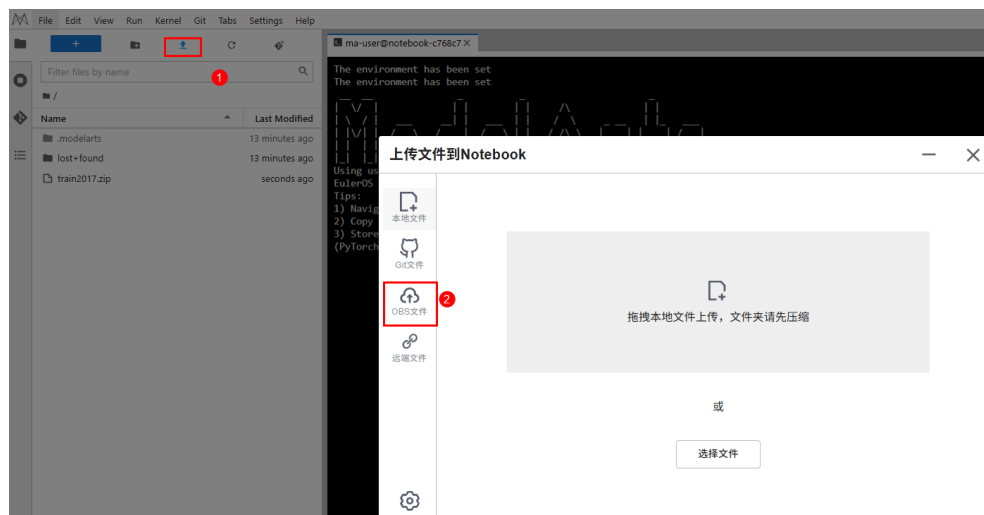
1.3.4 下载数据和模型代码到服务器

1.3.4.1 下载数据

创建保存数据的目录，并进入该目录，

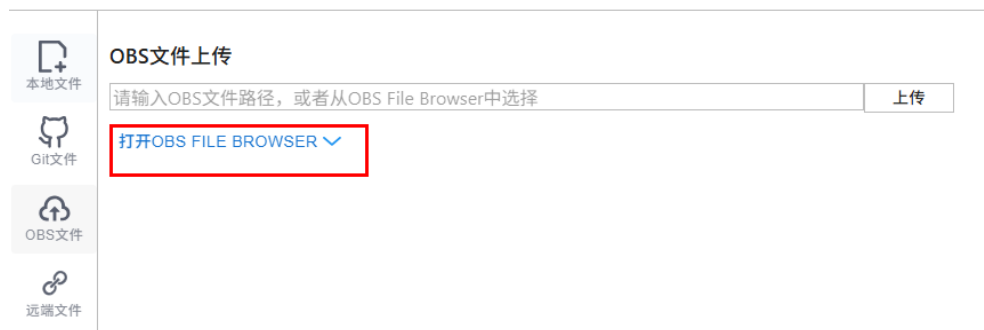
```
mkdir data && cd data
```

然后点击 Notebook 的 upload 按钮，



选择对应的 OBS 文件夹，

上传文件到Notebook





然后选择文件，将之前上传的数据全部下载到服务器中，



然后解压全部压缩数据。

```
unzip train2017.zip
unzip val2017.zip
unzip annotations_trainval2017.zip
```

1.3.4.2 下载 YOLOv5 代码文件

步骤 1 下载 gitee 仓代码，回到进入 Notebook 时的目录（例如：ma-user/work）。

```
git clone https://gitee.com/ascend/modelzoo-GPL.git
```

步骤 2 将 YOLOv5 代码拷贝到当前目录。

```
cp -r /home/ma-user/work/modelzoo-GPL/built-
in/PyTorch/Official/cv/object_detection/Yolov5_for_PyTorch_v6.0 ./
```

1.3.5 安装依赖包

1.3.5.1 安装 APEX

APEX 混合精度模块是一个集优化性能、精度收敛于一身的综合优化库，可以提供不同场景下的混合精度训练支持。

混合精度训练是在训练时混合使用单精度（float32）与半精度(float16)数据类型，将两者结合在一起，并使用相同的超参数实现了与 float32 几乎相同的精度。在迁移完成、训练开始之前，基于 NPU 硬件的架构特性，用户可以选择开启混合精度，用以提升模型的性能。

步骤 1 请确保已安装 PyTorch 框架且 setuptools 版本小于等于 65.7.0，若版本不符合条件，可使用以下命令安装。

通过 cd 命令回到进入 Notebook 时的目录（例如 ma-user/work），然后执行下面的命令。

```
pip3 install setuptools==65.7.0
```

步骤 2 获取昇腾适配的 APEX 源码。

```
git clone -b master https://gitee.com/ascend/apex.git
```

步骤 3 进入昇腾适配的 APEX 源码目录，执行命令编译生成二进制安装包。

```
cd apex
bash scripts/build.sh --python=3.10
```

步骤 4 执行如下命令进行安装。如果使用非 root 用户安装，需要在命令后加--user。

```
cd apex/dist/
pip3 uninstall apex      # 可选，若当前未安装 apex，可不执行
pip3 install apex-0.1+ascend-cp3x-cp3x-arch.whl    # x 指 Python 版本尾号，arch 指 CPU 架构
```

命令示例：

```
# 若用户在 x86 架构下安装，请将命令中文件包名中的“aarch64”改为“x86_64”。
pip3 install apex-0.1+ascend-cp310-cp310-linux_aarch64.whl
```

1.3.5.2 安装 YOLO 所需依赖

通过 cd 命令回到进入 Notebook 时的目录（例如 ma-user/work），执行如下命令。

```
cd YOLOv5_for_PyTorch_v6.0
pip install -r 2.1_requirements.txt
pip install seaborn
```

1.3.5.3 修改 numpy 版本

```
pip install numpy==1.23.4
```

1.3.6 处理数据集

1.3.6.1 COCO 2017 介绍

COCO（上下文中的常见物体）数据集是一个大规模物体检测、分割和字幕数据集。它旨在鼓励对各种物体类别进行研究，通常用于计算机视觉模型的基准测试。对于从事物体检测、分割和姿势估计任务的研究人员和开发人员来说，这是一个必不可少的数据集。

主要功能：

COCO 包含 330K 张图像，其中 200K 张图像具有用于对象检测、分割和图像描述任务的注释。

该数据集包含 80 个对象类别，包括常见的对象，如汽车、自行车和动物，以及更具体的类别，如雨伞、手提包和运动器材。

注释包括每个图像的对象边界框、分割掩码和描述。

COCO 提供了标准化的评估指标，例如用于目标检测的平均精度均值（mAP）和用于分割任务的平均召回率均值（mAR），使其适用于比较模型性能。

COCO 2017 是截至 2017 年 COCO 整理的数据库。

数据集结构：

COCO 数据集分为三个子集：

Train2017：此子集包含 118K 张图像，用于训练目标检测、分割和图像描述模型。

Val2017：此子集包含 5K 张图像，在模型训练期间用于验证。

Test2017：此子集包含 20K 张图像，用于测试和评估已训练的模型。此子集的 Ground Truth 标注不公开提供，结果将提交到 COCO 评估服务器以进行性能评估。

2017 Train/Val annotations：提供标注信息。

本实验主要使用前两者和标注信息。

1.3.6.2 标注文件转换

为什么需要转换？

标注文件格式差异：

COCO 数据集使用单一的 JSON 文件存储整个数据集的标注信息，包含五大部分：info(数据集元信息)、licenses(许可信息)、images(图片信息列表)、annotations(所有标注信息)和 categories(类别信息)。这种格式便于管理大型数据集，但不利于模型直接读取。相比之下，YOLO 模型要求每个图片对应一个独立的 TXT 文件，每行表示一个目标的标注信息，格式为“类别 ID x 中心 y 中心 宽 高”，其中所有坐标均为归一化值。

坐标表示方法差异：

COCO 标注使用边界框的左上角坐标(x_{min} , y_{min})和宽高(w , h)，而 YOLO 要求使用边界框的中心点坐标(x_{center} , y_{center})和宽高比例(w_{ratio} , h_{ratio})。这种差异直接影响模型对目标位置的理解，必须通过公式转换：

```
x_center = (x_min + w/2) / image_width
y_center = (y_min + h/2) / image_height
w_ratio = w / image_width
h_ratio = h / image_height
```

类别 ID 映射差异:

COCO 数据集的类别 ID 不连续(从 1 到 90)，而 YOLO 模型要求类别 ID 从 0 开始连续排列 90。例如，COCO 中"person"的 ID 是 1，"bicycle"是 2，"car"是 3，但在 YOLO 中应分别映射为 0、1、2，依此类推。如果不进行这种连续映射，模型在训练时会因无法正确识别类别而导致性能下降或错误。

步骤 1 确认数据完整，数据解压完成后，目录结构参考如下（根据自己保存目录确认）：

```

├── data #根目录
│   ├── train2017 #训练集图片，约 118287 张
│   ├── val2017 #验证集图片，约 5000 张
│   └── annotations #标注目录
│       ├── instances_train2017.json #对应目标检测、分割任务的训练集标注文件
│       ├── instances_val2017.json #对应目标检测、分割任务的验证集标注文件
│       ├── captions_train2017.json
│       ├── captions_val2017.json
│       ├── person_keypoints_train2017.json
│       └── person_keypoints_val2017.json

```

步骤 2 进入 YOLOv5 源码目录，将代码仓中 coco/coco2yolo.py 和 coco/coco_class.txt 拷贝到 data 数据集根目录。确认当前在进入 Notebook 时的目录（例如：ma-user/work）。

```

cp Yolov5_for_PyTorch_v6.0/coco/coco/coco2yolo.py ./data
cp Yolov5_for_PyTorch_v6.0/coco/coco/coco/coco_class.txt ./data

```

步骤 3 执行数据转换。

```

cd data/
python3 coco2yolo.py

```

运行上述脚本后，将在 coco 数据集根目录生成 train2017.txt 和 val2017.txt。

解析转换代码 coco2yolo.py。

```

from pycocotools.coco import COCO # 用于解析 COCO 格式的 JSON 标注文件
import shutil # 用于文件/目录的复制、删除等操作
import os # 用于操作系统相关功能（如创建目录、路径处理等）

# 边界框坐标转换函数 convert
def convert(size, box):
    dw = 1. / size[0] # 1/图像宽度（用于 x 坐标归一化）
    dh = 1. / size[1] # 1/图像高度（用于 y 坐标归一化）
    x = box[0] + box[2] / 2.0 # 计算边界框中心点 x 坐标（COCO 的 x_min + 宽/2）
    y = box[1] + box[3] / 2.0 # 计算边界框中心点 y 坐标（COCO 的 y_min + 高/2）
    w = box[2] # COCO 标注中的边界框宽度
    h = box[3] # COCO 标注中的边界框高度

```

```
x = x * dw # 中心点 x 坐标归一化 (除以图像宽度)
w = w * dw # 边界框宽度归一化
y = y * dh # 中心点 y 坐标归一化 (除以图像高度)
h = h * dh # 边界框高度归一化
return (x, y, w, h) # 返回 YOLO 格式的归一化坐标

# 类别映射函数 get_classes_and_index
def get_classes_and_index(path):
    D = {} # 用于存储“类别名称 → YOLO 类别 ID”的映射字典
    f = open(path) # 打开类别定义文件 (如 coco_class.txt)
    for line in f: # 逐行读取文件内容
        temp = line.rstrip().split(',', 2) # 以逗号分割行内容 (最多分割 2 次)
        print("temp[0]: " + temp[0] + "\n") # 打印类别 ID (调试用)
        print("temp[1]: " + temp[1] + "\n") # 打印类别名称 (调试用)
        D[temp[1]] = temp[0] # 建立映射: 类别名称 → ID (如 "person" → "0")
    return D # 返回映射字典

# 核心转换函数 coco2yolo
def coco2yolo(dataType): # dataType 为 'train2017' 或 'val2017', 表示处理训练集或验证集
    annFile = './annotations/instances_%.s.json' % dataType # COCO 标注文件路径 (如 instances_train2017.json)
    classes = get_classes_and_index('./coco_class.txt') # 加载类别映射字典

    # 创建图像存储目录 (若不存在)
    if not os.path.exists('./images'):
        os.makedirs('./images')

    # 为数据集图像创建符号链接 (避免复制大文件, 节省空间)
    # 链接路径: ./images/train2017 → 原始图像目录 (如 ./train2017)
    os.symlink(os.path.abspath(dataType), './images/%.s' % dataType)

    # 创建标签存储目录 (若存在则先删除再重建, 确保目录干净)
    if not os.path.exists('./labels/%.s' % dataType):
        os.makedirs('./labels/%.s' % dataType)
    else:
        shutil.rmtree('./labels/%.s' % dataType) # 删除已有目录及内容
        os.makedirs('./labels/%.s' % dataType)

    coco = COCO(annFile) # 初始化 COCO 对象, 加载标注文件
    list_file = open('%.s.txt' % dataType, 'w') # 创建文件 (如 train2017.txt), 用于存储所有图像的路径

    imgIds = coco.getImgIds() # 获取所有图像的 ID 列表
```

```

catIds = coco.getCatIds() # 获取所有类别的 ID 列表

# 遍历每张图像
for imgId in imgIds:
    objCount = 0 # 记录当前图像中有效目标的数量
    img = coco.loadImgs(imgId)[0] # 加载图像信息（如文件名、宽高）
    filename = img['file_name'] # 图像文件名（如 000000123456.jpg）
    width = img['width'] # 图像宽度（像素）
    height = img['height'] # 图像高度（像素）
    annIds = coco.getAnnIds(imgIds=imgId, catIds=catIds, iscrowd=None) # 获取当前图像的所有标注 ID

    # 遍历当前图像的所有标注
    for annId in annIds:
        anns = coco.loadAnns(annId)[0] # 加载标注信息（如边界框、类别 ID）
        catId = anns['category_id'] # 获取 COCO 的类别 ID
        cat = coco.loadCats(catId)[0]['name'] # 将 COCO 类别 ID 转换为类别名称（如
"person"）

        # 仅处理在 classes 字典中存在的类别（过滤不需要的类别）
        if cat in classes:
            objCount += 1 # 有效目标计数+1
            # 打开当前图像对应的 YOLO 标签文件（如 labels/train2017/000000123456.txt）
            out_file = open('labels/%s/%s.txt' % (dataType, filename[:-4]), 'a') # 'a'表示追加
            写入

            cls_id = classes[cat] # 获取 YOLO 的类别 ID（如"0"）
            box = anns['bbox'] # 获取 COCO 格式的边界框（[x_min, y_min, width, height]）
            size = [width, height] # 图像尺寸（宽，高）
            bb = convert(size, box) # 转换为 YOLO 格式的归一化坐标
            # 将类别 ID 和坐标写入标签文件（格式：class_id x_center y_center width height）
            out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) + '\n')
            out_file.close() # 关闭文件

            # 将当前图像的路径写入 list_file（供 YOLO 训练时读取所有图像）
            list_file.write('./images/%s/%s\n' % (dataType, filename))

list_file.close() # 关闭 list_file

# 主程序入口
if __name__ == '__main__':
    coco2yolo('train2017') # 转换训练集
    coco2yolo('val2017') # 转换验证集

```

1.3.6.3 在源码包根目录下配置数据集路径

```
cd ../Yolov5_for_PyTorch_v6.0/
```

```
mkdir datasets
ln -s coco_path ./datasets/coco # coco_path 为数据集实际路径，本次为/home/ma-user/work/data
```

1.4 训练模型

1.4.1 YOLOv5 介绍

YOLOv5 的架构由三个主要部分组成：

主干网络 (Backbone)：这是网络的主体。对于 YOLOv5，backbone 的设计使用了 CSPDarknet53 结构，是对先前版本中使用的 Darknet 架构的修改。

Neck：这部分连接 backbone 和 head。在 YOLOv5 中，SPPF（空间金字塔池化）和 PANet（Path Aggregation Network）结构。

头部：这部分负责生成最终输出。YOLOv5 使用了 YOLOv3 Head。

其结构的详细介绍，可以参考：

https://docs.ultralytics.com/zh/yolov5/tutorials/architecture_description/#1-model-structure。

1.4.2 启动训练

该模型支持单机单卡训练和单机 8 卡训练，本次实验使用单卡训练。同时，YOLOv5 也有不同的变体，例如：YOLOv5n、YOLOv5s、YOLOv5m、YOLOv5l 和 YOLOv5x，它们是改变了 V5 模型的深度和宽度，这个原理我们在课件中也讲到过，可以通过改变模型的深度和宽度来改变其性能，大致总结不同版本的 V5 使用场景如下。

版本	特点	适用场景
n	最小模型，参数量和计算量极低	移动设备、嵌入式设备、边缘计算（如手机、IoT设备）
s	小型模型，计算资源需求低	资源受限的设备（如树莓派、低功耗摄像头）
m	性能与资源的平衡	中等算力设备（如普通PC、轻量级服务器）
l	更深、更宽的网络，精度较高(标准版)	高性能服务器、需要高精度的场景（如工业检测）
x	最大模型，精度最高	高性能设备、追求极致精度的场景

YOLOv5 的不同版本通过 `depth_multiple`（深度因子）和 `width_multiple`（宽度因子）控制网络的深度（层数）和宽度（通道数）。以 l 版本为基准，调整不同版本的宽度和深度。例如 s 版本 `depth_multiple=0.33`，`width_multiple=0.5`。

在代码的 `models` 文件下，有个各种版本的模型结构文件。本实验使用 s（small）版本。打开 `yolov5s.yaml` 文件，查看 backbone 和 head 网络结构。

```
# Parameters
nc: 80 # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32
```

```
# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]], # 9
  ]
```

YOLOv5s 的检测头包含 特征融合（Neck）和 预测输出（Detect）两部分，采用 PANet（Path Aggregation Network）结构，实现多尺度特征融合。

```
# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]], # cat backbone P4
   [-1, 3, C3, [512, False]], # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]], # cat backbone P3
   [-1, 3, C3, [256, False]], # 17 (P3/8-small) 用于检测小目标

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]], # cat head P4
   [-1, 3, C3, [512, False]], # 20 (P4/16-medium) 用于检测中等目标

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]], # cat head P5
   [-1, 3, C3, [1024, False]], # 23 (P5/32-large) 用于检测大目标
```

```
[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
]
```

1.4.2.1 启动训练

启动单卡训练，在 yolo 源码根目录下，启动训练命令，

```
bash test/train_yolov5s_performance_1p.sh # yolov5s 1p_performance
```

启动训练后，控制台不继续显示信息，训练信息会保存在 log 日志中。训练过程需要等待很久，接下来分析训练启动脚本和训练代码。

1.4.2.2 启动训练的 shell 脚本解析

```
#!/bin/bash

# 定义训练任务的基础信息，通过命令行参数动态调整训练配置（如 sh train.sh --batch_size=32 --
data_path=/home/dataset）

# 网络名称（同目录名，用于标识训练任务），不需要修改
Network="yolov5s_ID4102_for_PyTorch_v6.0"

cur_path=`pwd`      # 当前目录路径，执行 pwd 获取
model_name=yolov5s  # 使用模型结构-5s
batch_size=64       # batch_size 大小
# 解析命令行参数（支持动态传入模型名称、批次大小、数据路径、是否启用 hf32 精度）
for para in $*
do
    if [[ $para == --model_name* ]];then
        model_name=`echo ${para#*=}` # 提取--model_name=xxx 中的 xxx 作为模型名称
    elif [[ $para == --batch_size* ]];then
        batch_size=`echo ${para#*=}` # 提取--batch_size=xxx 中的 xxx 作为批次大小
    elif [[ $para == --data_path* ]];then
        data_path=`echo ${para#*=}` # 提取--data_path=xxx 中的 xxx 作为数据集路径
    fi
done
# 校验并配置设备 ID（昇腾 NPU 需要指定具体设备），优先使用已有的 ASCEND_DEVICE_ID 环境变量（如
果已设置），若未设置，使用 local_rank 变量（分布式训练中常用）作为设备 ID，若均未配置，报错退出，
确保训练有可用的设备。
local_rank=0
if [ $ASCEND_DEVICE_ID ];then
    echo "device id is ${ASCEND_DEVICE_ID}"
elif [ ${local_rank} ];then
    export ASCEND_DEVICE_ID=${local_rank}
    echo "device id is ${ASCEND_DEVICE_ID}"
else
    "[Error] device id must be config"
    exit 1
fi
```

```
# #####指定训练脚本执行路径#####
# cd 到与 test 文件夹同层级目录下执行脚本，提高兼容性；test_path_dir 为包含 test 文件夹的路径
cur_path_last_dirname=${cur_path##*/}
if [ x"${cur_path_last_dirname}" == x"test" ]; then
    test_path_dir=${cur_path}
    cd ..
    cur_path=$(pwd)
else
    test_path_dir=${cur_path}/test
fi

# 创建 DeviceID 输出目录，不需要修改 if [ -d ${cur_path}/test/output/${ASCEND_DEVICE_ID} ]; # 路径为
# 当前目录/test/output/设备 ID，按设备 ID 区分不同设备的输出
then
    rm -rf ${cur_path}/test/output/${ASCEND_DEVICE_ID} #若目录已存在，先删除旧目录再重建，避免旧
    # 日志干扰新训练结果
    mkdir -p ${cur_path}/test/output/${ASCEND_DEVICE_ID} # 重新创建目录
else
    mkdir -p ${cur_path}/test/output/${ASCEND_DEVICE_ID}
fi

# 记录训练开始时间（用于计算总时长），不需要修改
start_time=$(date +%s) # 记录训练开始的时间戳（秒级）
echo "start_time: ${start_time}"

# 配置训练环境（区分平台场景和非平台场景）
check_etp_flag=$(env | grep etp_running_flag) # 检查是否在平台环境（如昇腾的 ETP 平台）
etp_flag=$(echo ${check_etp_flag}#*)
if [ x"${etp_flag}" != x"true" ]; then
    # 非平台场景：加载昇腾环境变量（如 NPU 驱动、PyTorch 适配插件等）
    source ${test_path_dir}/env_npu.sh
else
    # 平台场景：修改 COCO 数据集配置文件中的路径为用户指定的 data_path
    sed -i "s|./datasets/coco|${data_path}|g" ./data/coco.yaml
fi

# 执行训练命令（后台运行，输出重定向到日志）
python3 -u train.py --data ./data/coco.yaml \ # 启动训练文件 train.py，指定数据集配置文件（包含训练
/验证集路径、类别数等）
    --cfg yolov5s.yaml \ # 模型配置文件（定义网络结构、通道数等）
    --weights " \ # 不加载预训练权重（从头训练）
    --batch-size $batch_size \ # 批次大小（从参数传入）
    --device $ASCEND_DEVICE_ID \ # 指定昇腾设备 ID
```

```
--epochs 2 >
$cur_path/test/output/${ASCEND_DEVICE_ID}/train_${ASCEND_DEVICE_ID}.log 2>&1 & # 训练轮次（训练 2 轮）

wait # 等待后台训练进程完成

# 记录训练结束时间，计算总时长，不需要修改
end_time=$(date +%s)
echo "end_time: ${end_time}"
e2e_time=$(( $end_time - $start_time )) # 端到端训练总时长（秒）

# 从日志中提取最后一个迭代的 FPS（每秒处理图像数）
FPS=`grep -a 'FPS:' ${cur_path}/test/output/${ASCEND_DEVICE_ID}/train_${ASCEND_DEVICE_ID}.log \
    | awk 'END {print}' \ # 取最后一行包含 FPS 的记录
    | awk -F "[" '{print $5}' \ # 按 "[" 分割，取第 5 段
    | awk -F "]" '{print $1}' \ # 按 "]" 分割，取第 1 段
    | awk -F ":" '{print $2}'` # 按 ":" 分割，取 FPS 值

# 打印性能数据，不需要修改
echo "Final Performance images/sec : $FPS"
echo "E2E Training Duration sec : $e2e_time"

# #稳定性精度看护结果汇总
#训练用例信息，不需要修改
BatchSize=${batch_size}
DeviceType=`uname -m` # 设备架构（如 aarch64 或 x86_64）
CaseName=${Network}_bs${BatchSize}_${RANK_SIZE}'p'_'perf' # 用例名称（包含网络、批次、设备数等）

##获取性能数据，不需要修改
#单迭代训练时长
TrainingTime=`awk 'BEGIN{printf "%.2f\n", '${batch_size}'*1000/'${FPS}'}'`

#关键信息打印到${CaseName}.log 中，不需要修改，打印前面定义的数值，例如 FPS 值、训练时长等
echo "Network = ${Network}" > $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "RankSize = ${RANK_SIZE}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "BatchSize = ${BatchSize}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "DeviceType = ${DeviceType}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "CaseName = ${CaseName}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "ActualFPS = ${FPS}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "TrainingTime = ${TrainingTime}" >>
$cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
echo "E2E TrainingTime = ${e2e_time}" >> $cur_path/test/output/${ASCEND_DEVICE_ID}/${CaseName}.log
# 清理数据集缓存文件（避免下次训练使用旧缓存）
rm -rf $data_path/labels/*.cache
```

1.4.2.3 解析训练代码

启动训练的 shell 文件中指定了训练文件是 train.py，接下来详细分析 train.py 中内容。

步骤 1 导入依赖库

在这段代码中，导入的库可分为基础工具库、深度学习框架核心库、YOLOv5 自定义核心模块和硬件适配库四类，下面选取较为重要部分注释。

```
import argparse # 用于解析命令行参数，处理训练时的配置输入（如批次大小、训练轮次等）
import logging # 提供日志记录功能，用于输出训练过程中的关键信息、警告和错误
import math # 提供数学运算支持，用于训练中的各种数值计算（如学习率调整、损失计算等）
import os
import random
import sys
import time
import yaml # 解析 YAML 格式文件，加载数据集配置（如 coco.yaml）和模型结构配置（如 yolov5s.yaml）
from copy import deepcopy
from pathlib import Path
from tqdm import tqdm # 提供随机数生成和数组运算支持，用于数据增强、初始化等场景

import numpy as np
import torch
if torch.__version__ >= '1.8': # 判断 torch 版本，如果大于 1.8，则导入 torch_npu，确保 Pytorch 版本大于等于 1.8
    import torch_npu
    torch.npu.config.allow_internal_format = True # 是否使用私有格式，设置为 True 时允许使用私有格式，设置为 False 时，不允许申请任何私有格式的 tensor，避免了适配层出现私有格式流通。
import torch.distributed as dist # 分布式计算
import torch.nn as nn
from torch.cuda import amp # 支持混合精度训练
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.optim import Adam, SGD, lr_scheduler

import apex # 混合精度模块
from apex import amp
try:
    from torch_npu.utils.profiler import Profile
except:
    print("Profile not in torch_npu.utils.profiler now.. Auto Profile disabled.", flush=True)
    class Profile:
        def __init__(self, *args, **kwargs):
            pass

        def start(self):
            pass
```

```
def end(self):
    pass

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0] # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT)) # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative

import val # for end-of-epoch mAP
from models.experimental import attempt_load # 加载模型权重
from models.yolo import Model # 加载模型结构
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path, labels_to_image_weights, init_seeds, \
    strip_optimizer, get_latest_run, check_dataset, check_git_status, check_img_size, check_requirements, \
    check_file, check_yaml, check_suffix, print_args, print_mutation, set_logging, one_cycle, colorstr, \
    methods
from utils.downloads import attempt_download
from utils.loss import ComputeLoss
from utils.plots import plot_labels, plot_evolve
from utils.torch_utils import EarlyStopping, ModelEMA, de_parallel, intersect_dicts, select_device, \
    torch_distributed_zero_first
from utils.loggers.wandb.wandb_utils import check_wandb_resume
from utils.metrics import fitness
from utils.loggers import Loggers
from utils.callbacks import Callbacks
```

步骤 2 定义训练函数

train 函数是 YOLOv5 训练流程的核心，负责参数解析、模型构建、数据加载、训练循环、性能监控、权重保存等全流程逻辑。

```
def train(hyp, # path/to/hyp.yaml or hyp dictionary 超参数文件路径或字典
          opt, # 训练选项/参数
          device, # 训练设备-npu
          callbacks # 回调函数
        ):
    # 1. 从 opt 中解包核心训练参数（解包后变量与 opt 属性一一对应，简化后续调用）
    # 解释：
    # save_dir: 训练结果（权重、日志）保存目录（Path 类型，方便路径操作）
    # epochs: 总训练轮次
    # batch_size: 单设备批次大小
    # weights: 预训练权重路径（如 yolov5s.pt）
    # single_cls: 是否单类别训练（True/False）
    # evolve: 是否开启超参进化（用于超参优化）
```

```
# data: 数据集配置文件路径 (如 coco.yaml)
# cfg: 模型结构配置文件路径 (如 yolov5s.yaml)
# resume: 是否断点续训 (True/False)
# noval: 是否跳过验证 (True/False)
# nosave: 是否不保存权重 (True/False, 仅测试用)
# workers: 数据加载线程数
# freeze: 需要冻结的模型层数 (如 freeze=10 表示冻结前 10 层)
save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg, resume, noval, nosave, workers,
freeze, = \
    Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights, opt.single_cls, opt.evolve, opt.data,
opt.cfg, \
    opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze

# 2. 创建权重保存目录
# Directories
w = save_dir / 'weights' # weights dir
(w.parent if evolve else w).mkdir(parents=True, exist_ok=True) # make dir
last, best = w / 'last.pt', w / 'best.pt'

# 3. 加载并解析超参数
# Hyperparameters
if isinstance(hyp, str): # 判断 hyp 是否为路径字符串 (而非字典)
    with open(hyp, errors='ignore') as f: # 打开超参 YAML 文件 (忽略编码错误)
        hyp = yaml.safe_load(f) # load hyps dict
# 打印超参 (带颜色高亮)
# 解释: 超参包括学习率 (lr0)、动量 (momentum)、损失权重 (box/cls/obj) 等,
# 控制训练过程的核心参数
LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))

# 4. 保存超参和命令行参数到本地 (存档用, 确保训练可复现)
# Save run settings
with open(save_dir / 'hyp.yaml', 'w') as f:
    yaml.safe_dump(hyp, f, sort_keys=False)
with open(save_dir / 'opt.yaml', 'w') as f:
    yaml.safe_dump(vars(opt), f, sort_keys=False)
data_dict = None

# 5. 初始化日志器 (仅主进程处理, 避免分布式场景下多设备重复日志)
# Loggers
if RANK in [-1, 0]:
    loggers = Loggers(save_dir, weights, opt, hyp, LOGGER) # loggers instance
    if loggers.wandb:
        data_dict = loggers.wandb.data_dict
```

```
        if resume:
            weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp

        # Register actions
        for k in methods(loggers):
            callbacks.register_action(k, callback=getattr(loggers, k))

# 6. 数据集配置与校验（分布式场景下主进程先处理，其他进程等待）
# Config
plots = not evolve # create plots
npu = device.type != 'cpu' # 判断是否使用 NPU（设备类型非 CPU 即为 NPU/GPU，此处适配昇腾 NPU）
init_seeds(1 + RANK) # 初始化随机种子（不同设备用不同种子，保证分布式训练一致性）
with torch_distributed_zero_first(LOCAL_RANK):
    data_dict = data_dict or check_dataset(data) # check if None，加载数据集配置
# check_dataset(data): 解析 data.yaml，返回数据集字典（含 train/val 路径、nc、names 等）
# 处理类别名（单类别训练时强制类别名为['item']，否则用数据集配置的 names）
train_path, val_path = data_dict['train'], data_dict['val'] # 提取训练集、验证集路径
# 确定类别数（单类别训练则 nc=1，否则用数据集配置的 nc）
nc = 1 if single_cls else int(data_dict['nc']) # number of classes
# class names
names = ['item'] if single_cls and len(data_dict['names']) != 1 else data_dict['names']
# 校验：类别名数量必须等于类别数
assert len(names) == nc, f'{len(names)} names found for nc={nc} dataset in {data}' # check
is_coco = data.endswith('coco.yaml') and nc == 80 # COCO dataset

# 7. 模型构建与预训练权重加载
# Model
check_suffix(weights, '.pt') # check weights，检查权重文件后缀是否为.pt（避免加载非权重文件）
pretrained = weights.endswith('.pt') # 判断是否使用预训练权重（路径以.pt 结尾）
if pretrained:
    with torch_distributed_zero_first(LOCAL_RANK):
        weights = attempt_download(weights) # download if not found locally
        ckpt = torch.load(weights, map_location=device) # load checkpoint
        model = Model(cfg or ckpt['model'].yaml, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)
# create
exclude = ['anchor'] if (cfg or hyp.get('anchors')) and not resume else [] # exclude keys
csd = ckpt['model'].float().state_dict() # checkpoint state_dict as FP32
csd = intersect_dicts(csd, model.state_dict(), exclude=exclude) # intersect
model.load_state_dict(csd, strict=False) # load
LOGGER.info(f'Transferred {len(csd)}/{len(model.state_dict())} items from {weights}') # report
else:
    model = Model(cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device) # create
```

```
# 8. 模型层冻结（冻结早期层，仅训练后期层，适用于微调场景）
# Freeze
freeze = [f'model.{x}.' for x in range(freeze)] # layers to freeze
for k, v in model.named_parameters(): # 遍历模型所有可训练参数
    v.requires_grad = True # train all layers
    if any(x in k for x in freeze):
        print(f'freezing {k}')
        v.requires_grad = False # 冻结该参数（反向传播时不更新）

# 9. 优化器配置（适配昇腾 NPU，支持梯度累积、参数分组）
# Optimizer
nbs = 64 # nominal batch size
accumulate = max(round(nbs / batch_size), 1) # accumulate loss before optimizing
if not opt.noscale:
    hyp['weight_decay'] *= batch_size * accumulate / nbs # scale weight_decay
    LOGGER.info(f"Scaled weight_decay = {hyp['weight_decay']}")

# 将模型参数分为 3 组（不同组用不同优化策略）
g0, g1, g2 = [], [], [] # optimizer parameter groups
for v in model.modules():
    if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter): # bias
        g2.append(v.bias)
    if isinstance(v, nn.BatchNorm2d): # weight (no decay)
        g0.append(v.weight)
    elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter): # weight (with decay)
        g1.append(v.weight)

# 选择优化器（适配昇腾 NPU 的融合优化器，提升训练速度）
optimizer = None
if opt.adam:
    optimizer = Adam(g0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999)) # adjust beta1 to
momentum
else:
    if os.environ["use_amp"] == "apex": # 若使用 Apex 混合精度
        optimizer = apex.optimizers.NpuFusedSGD(g0, lr=hyp['lr0'], momentum=hyp['momentum'],
nesterov=True)
    elif os.environ["use_amp"] in ["native", "false"]:
        optimizer = torch_npu.optim.NpuFusedSGD(g0, lr=hyp['lr0'],
momentum=hyp['momentum'], nesterov=True)

# 为优化器添加剩余参数组（g1 加权重衰减，g2 无衰减）
optimizer.add_param_group({'params': g1, 'weight_decay': hyp['weight_decay']}) # add g1 with
weight_decay
optimizer.add_param_group({'params': g2}) # add g2 (biases)
LOGGER.info(f"{colorstr('optimizer:')} {type(optimizer).__name__} with parameter groups "
f"{len(g0)} weight, {len(g1)} weight (no decay), {len(g2)} bias")

# 删除临时参数组列表，释放内存
```

```
del g0, g1, g2

# 10. 混合精度训练配置
if os.environ["use_amp"] == "apex":
    model, optimizer = amp.initialize(model, optimizer, loss_scale=128, combine_grad=True,
                                      combine_ddp=True if npu and RANK != -1 else False)
elif os.environ["use_amp"] == "native":
    scaler = torch.npu.amp.GradScaler(dynamic=False, init_scale=128)

# 11. 学习率调度器配置
# Scheduler
if opt.linear_lr:
    lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf'] # linear
else:
    lf = one_cycle(1, hyp['lrf'], epochs) # cosine 1->hyp['lrf']
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf) # plot_lr_scheduler(optimizer,
scheduler, epochs)

# 12. EMA (指数移动平均) 初始化
# EMA
ema = ModelEMA(model) if RANK in [-1, 0] else None

# 13. 断点续训处理
# Resume
start_epoch, best_fitness = 0, 0.0
if pretrained:
    # Optimizer
    if ckpt['optimizer'] is not None:
        optimizer.load_state_dict(ckpt['optimizer'])

    # EMA
    if ema and ckpt.get('ema'):
        ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
        ema.updates = ckpt['updates']

    # Epochs
    start_epoch = ckpt['epoch'] + 1
    if resume:
        assert start_epoch > 0, f'{weights} training to {epochs} epochs is finished, nothing to
resume.'
    if epochs < start_epoch:
        LOGGER.info(f'{weights} has been trained for {ckpt['epoch']} epochs. Fine-tuning for
{epochs} more epochs.')
        epochs += ckpt['epoch'] # finetune additional epochs

del ckpt, csd
```

```
# 14. 输入图像尺寸校验 (YOLOv5 要求尺寸是模型最大步幅的倍数)
# Image sizes
gs = max(int(model.stride.max()), 32) # grid size (max stride)
nl = model.model[-1].nl # number of detection layers (used for scaling hyp['obj'])
imgsz = check_img_size(opt.imgsz, gs, floor=gs * 2) # verify imgsz is gs-multiple


# 15. 分布式场景下启用 SyncBatchNorm (同步各设备 BN 层统计量, 提升精度)
# SyncBatchNorm
if opt.sync_bn and npu and RANK != -1:
    model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
    LOGGER.info('Using SyncBatchNorm()')


# 16. 创建训练集加载器 (支持数据增强、缓存、图像权重采样)
# Trainloader
train_loader, dataset = create_dataloader(
    train_path, # 训练集路径
    imgsz, # 训练图像尺寸
    batch_size // WORLD_SIZE, # 单设备批次大小 (分布式场景下总 batch_size=单设备*设备数)
    gs, # 模型最大步幅 (用于尺寸对齐)
    single_cls, # 是否单类别
    hyp=hyp, # 超参
    augment=True, # 启用训练数据增强 (如马赛克、翻转、色域抖动)
    cache=opt.cache, # 是否缓存数据到内存/磁盘 (加速后续加载)
    rect=opt.rect, # 是否用矩形训练 (减少 padding, 提升速度)
    rank=LOCAL_RANK, # 本地设备编号 (分布式场景)
    workers=workers, # 数据加载线程数
    image_weights=opt.image_weights, # 是否按图像权重采样 (平衡类别分布)
    quad=opt.quad, # 是否用四马赛克增强 (提升小目标检测性能)
    prefix=colorstr('train: ') # 日志前缀 (区分训练/验证加载器)

# 校验数据集标签的最大类别 (确保标签类别不超过 nc)
mlc = int(np.concatenate(dataset.labels, 0)[: , 0].max()) # max label class
nb = len(train_loader) # number of batches
assert mlc < nc, f'Label class {mlc} exceeds nc={nc} in {data}. Possible class labels are 0-{nc - 1}'
# 17. 主进程创建验证集加载器+预处理 (非续训时绘制标签分布、检查锚点), 参数同上类似
# Process 0
if RANK in [-1, 0]:
    val_loader = create_dataloader(
        val_path,
        imgsz,
        batch_size // WORLD_SIZE * 2,
        gs,
        single_cls,
```

```
hyp=hyp,
cache=None if noval else opt.cache,
rect=True,
rank=-1,
workers=workers,
pad=0.5,
prefix=colorstr('val: '))[0]

if not resume: # 非续训（首次训练）
    # 提取所有训练集标签（用于绘制标签分布、初始化偏置）
    labels = np.concatenate(dataset.labels, 0)
    # c = torch.tensor(labels[:, 0]) # classes
    # cf = torch.bincount(c.long(), minlength=nc) + 1. # frequency
    # model._initialize_biases(cf.to(device))
    if plots:
        plot_labels(labels, names, save_dir)

    # Anchors 自动锚点适配（若未禁用 noautoanchor）
    if not opt.noautoanchor:
        check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)
    model.half().float() # pre-reduce anchor precision

callbacks.run('on_pretrain_routine_end')

# 18. 分布式数据并行（DDP）包装模型（非 CPU/非 FP32 场景）
# DDP mode
if (not npu or opt.FP32) and RANK != -1:
    model = DDP(model, device_ids=[LOCAL_RANK], output_device=LOCAL_RANK,
broadcast_buffers=False)

# 19. 附加关键参数到模型（方便后续损失计算、日志记录）
# Model parameters
hyp['box'] *= 3. / nl # scale to layers 缩放 box 损失权重（3 个检测头，平均分配）
hyp['cls'] *= nc / 80. * 3. / nl # scale to classes and layers 缩放 cls 损失权重
hyp['obj'] *= (imgsz / 640) ** 2 * 3. / nl # scale to image size and layers
hyp['label_smoothing'] = opt.label_smoothing
model.nc = nc # attach number of classes to model
model.hyp = hyp # attach hyperparameters to model
model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device) * nc # attach class
weights
model.names = names

# 20. 训练前初始化（时间、损失、性能指标）
# Start training
```

```
t0 = time.time()
nw = max(round(hyp['warmup_epochs'] * nb), 1000) # number of warmup iterations, max(3
epochs, 1k iterations)
# nw = min(nw, (epochs - start_epoch) / 2 * nb) # limit warmup to < 1/2 of training
last_opt_step = -1
maps = np.zeros(nc) # mAP per class
results = (0, 0, 0, 0, 0, 0, 0) # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
scheduler.last_epoch = start_epoch - 1 # do not move
stopper = EarlyStopping(patience=opt.patience)
compute_loss = ComputeLoss(model) # init loss class
# 打印训练配置信息（图像尺寸、线程数、日志路径、总轮次）
LOGGER.info(f'Image sizes {imgsz} train, {imgsz} val\n'
            f'Using {train_loader.num_workers} dataloader workers\n'
            f'Logging results to {colorstr('bold', save_dir)}\n'
            f'Starting training for {epochs} epochs...')

# 21. Epoch 级训练循环（遍历每个训练轮次）
for epoch in range(start_epoch, epochs):
    # epoch -----
    model.train()

    # Update image weights (optional, single-GPU only)
    if opt.image_weights:
        cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc # class weights
        iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw) # image weights
        # 按图像权重随机采样（权重高的图像被选中概率高）
        dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n) # rand
weighted idx

    # Update mosaic border (optional) 随机调整马赛克边界（可选，增强数据多样性）
    # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
    # dataset.mosaic_border = [b - imgsz, -b] # height, width borders

    初始化损失变量（4个损失：box、obj、cls、total）
    mloss = torch.zeros(4, device=device) # mean losses
    if RANK != -1:
        train_loader.sampler.set_epoch(epoch)
    optimizer.zero_grad()
    start_time = time.time()
    d_l = time.time()
    profile = Profile(start_step=int(os.getenv('PROFILE_START_STEP', 10)),
                      profile_type=os.getenv('PROFILE_TYPE'))

    # 22. Batch 级训练循环（遍历每个训练批次）
    for i, (imgs, targets, paths, _) in enumerate(train_loader):
```

```

# batch -----
t_time = time.time()
d_time = t_time - d_l
ni = i + nb * epoch # number integrated batches (since train start)
imgs = imgs.to(device, non_blocking=True).float() / 255.0 # uint8 to float32, 0-255 to 0.0-1.0

# Warmup 前 nw 次迭代，逐步调整学习率和动量
if ni <= nw:
    xi = [0, nw] # x interp 迭代范围
    # compute_loss.gr = np.interp(ni, xi, [0.0, 1.0]) # iou loss ratio (obj_loss = 1.0 or iou)
    accumulate = max(1, np.interp(ni, xi, [1, nbs / batch_size])).round()
    # 调整每个参数组的学习率和动量
    for j, x in enumerate(optimizer.param_groups):
        # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
        x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0, x['initial_lr'] *
lf(epoch)])
        if 'momentum' in x:
            x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'],
hyp['momentum']]))

# Multi-scale 多尺度训练（随机调整图像尺寸，提升模型泛化能力）
if opt.multi_scale:
    sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs # size
    sf = sz / max(imgs.shape[2:]) # scale factor
    if sf != 1:
        ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]] # new shape (stretched to
gs-multiple)
        imgs = nn.functional.interpolate(imgs, size=ns, mode='bilinear',
align_corners=False)

# Forward 前向传播
profile.start() # 启动性能分析
if os.environ["use_amp"] == "native":
    with torch.npu.amp.autocast():
        pred = model(imgs) # forward
        # loss scaled by batch_size
        loss, loss_items = compute_loss(pred, targets.to(device))
else:
    pred = model(imgs)
    loss, loss_items = compute_loss(pred, targets.to(device)) # loss scaled by batch_size

# 分布式场景：缩放损失（DDP 模式下梯度会自动平均，故先乘设备数）
if RANK != -1:
    loss *= WORLD_SIZE # gradient averaged between devices in DDP mode
if opt.quad: 四马赛克增强：损失乘 4（因四马赛克是 4 张图合成 1 张，损失需对应 4 倍）
    loss *= 4.

```

```
# Backward 反向传播（计算梯度）
if opt.FP32 or os.getenv('ALLOW_HF32'):
    loss.backward()
elif os.environ["use_amp"] == "apex":
    with amp.scale_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
elif os.environ["use_amp"] == "native":
    scaler.scale(loss).backward()

# Optimize 梯度累积优化（累积 accumulate 批梯度后再更新参数）
if ni - last_opt_step >= accumulate:
    if os.environ["use_amp"] == "native":
        scaler.step(optimizer)
        scaler.update()
    else:
        optimizer.step()
    optimizer.zero_grad()
    if ema:
        if device.type == 'npu':
            params_fp32_fused = None
            if os.environ["use_amp"] == "apex":
                params_fp32_fused = optimizer.get_model_combined_params()
            elif os.environ["use_amp"] in ["native", "false"]:
                params_fp32_fused = optimizer.get_combined_params()
            ema.update(model, device, params_fp32_fused[0])
        else:
            ema.update(model, device)
    last_opt_step = ni
profile.end()

# 计算训练速度（FPS）和时间
if i <= 10: # 前 11 个批次：计算平均时间（含初始化耗时）
    sum_time = (time.time() - start_time) / (i + 1)
    if i == 10:
        start_time = time.time()
else: # 第 11 个批次后：计算真实训练时间（排除前 10 个批次的初始化耗时）
    sum_time = (time.time() - start_time) / (i - 10)
ptime = time.time() - d_l

# Log 主进程打印实时训练日志（epoch、批次、设备、FPS、损失等）
if RANK in [-1, 0]:
    mloss = (mloss * i + loss_items) / (i + 1) # update mean losses
    print(
        'Epoch:[%2g][%4g/%4g][%s][FPS:%3.1f][mTime:%3.3f][pTime:%3.3f][dTime:%3.3f]
        IoU:%.3f objectness:%.3f classification:%.3f totalLoss:%.3f' % (
```

```

        epoch, i, nb, device, opt.batch_size / sum_time, sum_time, ptime, d_time, *mloss))
    d_l = time.time() # 更新数据加载开始时间（为下一批次准备）
    # end batch -----

# 23. 每个 Epoch 结束后处理
# Scheduler
scheduler.step() # 更新学习率（按调度器策略）

# save every epoch 每个 Epoch 保存一次权重
if RANK in [-1, 0]:
    ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'names', 'stride', 'class_weights'])
    final_epoch = epoch + 1 == epochs
    ckpt = {
        'epoch': epoch,
        'model': ema.ema.module if hasattr(ema, 'module') else ema.ema,
        'optimizer': None if final_epoch else optimizer.state_dict()
    }
    torch.save(ckpt, 'yolov5.pt')
    print('ckpt saved...')

    # Stop DDP TODO: known issues shttps://github.com/ultralytics/yolov5/pull/4576
    # stop = stopper(epoch=epoch, fitness=fi)
    # if RANK == 0:
    #     dist.broadcast_object_list([stop], 0) # broadcast 'stop' to all ranks

# Stop DPP
# with torch_distributed_zero_first(RANK):
# if stop:
#     break # must break all DDP ranks

# end epoch -----
# end training -----

# 24. 训练结束处理
if RANK in [-1, 0]:
    # 计算并打印总训练耗时（小时）
    LOGGER.info(f'\n{epoch - start_epoch + 1} epochs completed in {(time.time() - t0) / 3600:.3f}
hours.')

    torch.npu.empty_cache() # 清空 NPU 缓存（释放未使用的张量内存）
    return results # 返回验证集性能结果（供后续分析）

```

以上是训练的代码，下面定义配置参数的函数

```

def parse_opt(known=False):
    # 模型与配置文件参数
    parser = argparse.ArgumentParser()

```

```
# 初始权重路径（默认使用 yolov5s.pt 预训练权重）
parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
# 模型结构配置文件（如 yolov5s.yaml，默认空表示从权重中加载配置）
parser.add_argument('--cfg', type=str, default="", help='model.yaml path')
# 数据集配置文件
parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
# 超参数配置文件
parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch.yaml', help='hyperparameters path')

# 训练控制参数 控制训练的基本流程：轮次、批次大小、图像尺寸、续训、验证策略等。
parser.add_argument('--epochs', type=int, default=300)
parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
parser.add_argument('--rect', action='store_true', help='rectangular training')
parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
parser.add_argument('--noval', action='store_true', help='only validate final epoch')
parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
parser.add_argument('--bucket', type=str, default="", help='gsutil bucket')

# 数据增强与加载参数
parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
parser.add_argument('--device', default="", help='cuda device, i.e. 0 or 0,1,2,3 or cpu') # 训练设备
parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')

# 优化器与学习率参数
parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
parser.add_argument('--workers', type=int, default=24, help='maximum number of dataloader workers')

# 输出与早停参数
parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
parser.add_argument('--name', default='exp', help='save to project/name')
parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--quad', action='store_true', help='quad dataloader')
```

```
parser.add_argument('--linear-lr', action='store_true', help='linear LR')
parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
parser.add_argument('--freeze', type=int, default=0, help='Number of layers to freeze. backbone=10, all=24')
parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
parser.add_argument('--noscale', action='store_true', help='do not scale weight decay')

# Weights & Biases arguments
parser.add_argument('--entity', default=None, help='W&B: Entity')
parser.add_argument('--upload_dataset', action='store_true', help='W&B: Upload dataset as artifact table')
parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set bounding-box image logging interval')
parser.add_argument('--artifact_alias', type=str, default='latest', help='W&B: Version of dataset artifact to use')

# amp 精度控制参数 (适配昇腾 NPU)
parser.add_argument('--native_amp', action='store_true', help='use torch amp')
#FP32
parser.add_argument('--FP32', action='store_true', help='use FP32')

# 参数解析与混合精度配置
opt = parser.parse_known_args()[0] if known else parser.parse_args()

if opt.FP32 or os.getenv('ALLOW_HF32'):
    os.environ['use_amp'] = 'false'
elif opt.native_amp:
    os.environ['use_amp'] = 'native'
else:
    os.environ['use_amp'] = 'apex'

return opt # 返回解析后的参数对象
```

接下来是定义主函数和启动函数，主要是一些逻辑设置，因此不展开解析。

启动训练后，可以在 test/output/0 文件夹下查看 log 信息—train_0.log，首先输出模型结构和总参数量，

```

from n params module arguments
0 -1 1 3520 models.common.Conv [3, 32, 6, 2, 2]
1 -1 1 18560 models.common.Conv [32, 64, 3, 2]
2 -1 1 18816 models.common.C3 [64, 64, 1]
3 -1 1 73984 models.common.Conv [64, 128, 3, 2]
4 -1 2 115712 models.common.C3 [128, 128, 2]
5 -1 1 295424 models.common.Conv [128, 256, 3, 2]
6 -1 3 625152 models.common.C3 [256, 256, 3]
7 -1 1 1180672 models.common.Conv [256, 512, 3, 2]
8 -1 1 1182720 models.common.C3 [512, 512, 1]
9 -1 1 656896 models.common.SPPF [512, 512, 5]
10 -1 1 131584 models.common.Conv [512, 256, 1, 1]
11 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
12 [-1, 6] 1 0 models.common.Concat [1]
13 -1 1 361984 models.common.C3 [512, 256, 1, False]
14 -1 1 33024 models.common.Conv [256, 128, 1, 1]
15 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
16 [-1, 4] 1 0 models.common.Concat [1]
17 -1 1 90880 models.common.C3 [256, 128, 1, False]
18 -1 1 147712 models.common.Conv [128, 128, 3, 2]
19 [-1, 14] 1 0 models.common.Concat [1]
20 -1 1 296448 models.common.C3 [256, 256, 1, False]
21 -1 1 590336 models.common.Conv [256, 256, 3, 2]
22 [-1, 10] 1 0 models.common.Concat [1]
23 -1 1 1182720 models.common.C3 [512, 512, 1, False]
24 [17, 20, 23] 1 229245 models.yolo.Detect [80, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
Model Summary: 270 layers, 7235389 parameters, 7235389 gradients

```

打印完信息，设置一些超参数和缓存数据数据，中间需要等待。结束后启动训练，直至结束，训练数据类似下图，

```

Starting training for 2 epochs...
.....[W compiler_depend.ts:82] Warning: The operator of add is executed, Currently High Accuracy but Low Performance OP with 64-bit has been used, Please Do
Some Cast at Python Functions with 32-bit for Better Performance! (function operator())
.....[W compiler_depend.ts:87] Warning: [Check][offset] Check input storage_offset[%d] = 0 failed, result is untrustworthy33600 (function operator())
.....group num: 3
...Epoch: 0 [ 0/1848][npu:0][FPS:0.1][mTime:610.563][pTime:610.563][dTime:10.035] IoU:0.110 objectness:0.076 classification:0.106 totalLoss:0.292
...Epoch: 0 [ 1/1848][npu:0][FPS:0.2][mTime:323.348][pTime:29.888][dTime:0.001] IoU:0.110 objectness:0.073 classification:0.106 totalLoss:0.289
Epoch: 0 [ 2/1848][npu:0][FPS:0.3][mTime:215.633][pTime:0.203][dTime:0.001] IoU:0.110 objectness:0.074 classification:0.106 totalLoss:0.290
Epoch: 0 [ 3/1848][npu:0][FPS:0.4][mTime:161.774][pTime:0.197][dTime:0.001] IoU:0.110 objectness:0.073 classification:0.106 totalLoss:0.289
Epoch: 0 [ 4/1848][npu:0][FPS:0.5][mTime:129.460][pTime:0.201][dTime:0.001] IoU:0.110 objectness:0.071 classification:0.106 totalLoss:0.287
Epoch: 0 [ 5/1848][npu:0][FPS:0.6][mTime:107.917][pTime:0.201][dTime:0.001] IoU:0.110 objectness:0.072 classification:0.106 totalLoss:0.288
Epoch: 0 [ 6/1848][npu:0][FPS:0.7][mTime:92.529][pTime:0.200][dTime:0.000] IoU:0.109 objectness:0.072 classification:0.106 totalLoss:0.287

.....

Epoch: 1 [1841/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.200][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1842/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.194][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1843/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.199][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1844/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.199][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1845/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.200][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1846/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.200][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
Epoch: 1 [1847/1848][npu:0][FPS:315.5][mTime:0.203][pTime:0.200][dTime:0.000] IoU:0.073 objectness:0.085 classification:0.072 totalLoss:0.230
ckpt saved...

2 epochs completed in 0.402 hours.

```

训练完 checkpoint 会保存在当前目录下。

1.5 思考题

训练完模型后，如何进行评测，即检测训练好的模型检测物体的精确度？

解析：代码中给出了每个版本模型的预测 shell 脚本和对应的 val.py 文件，例如执行以下命令启动预测：

```
bash ./test/train_yolov5s_eval.sh #在线推理
```

模型训练脚本参数说明如下：

公共参数：

```

--conda_name           //conda 名，不传默认没有，传了进入 conda 执行
--data                 //数据集路径
--workers              //加载数据进程数
--batch-size           //训练批次大小
--epochs               //重复训练次数
--weights              //初始权重路径
--rect                 //矩形训练

```

```
--nosave                //保存最后一个权重
--noval                 //验证最后一个 epoch
--artifact_alias        //数据集版本
--save-period           //权重保存
--native_amp            //使用 torch amp 进行混合精度训练，如不配置默认使用 apex
--half                 //eval 执行脚本中参数，如配置默认使用混合精度计算
```

--data_path 参数填写数据集路径，需写到数据集的一级目录。

1.6 本章总结

本章讲解了 YOLOv5 的训练流程，主要包含数据集的获取，数据的格式处理，训练主要代码解析，训练启动脚本参数解析等内容。

华为认证 AI 系列教程

HCIP-AI-Model Developer

Qwen3-4B微调

实验指导手册

版本：2.0



华为技术有限公司

版权所有 ©【2025】【华为技术有限公司】。保留一切权利。

未经华为事先书面许可，任何人不得对本资料进行复制、修改、整合、改编、翻译，也不得将本资料或其任何部分或基于本资料的衍生作品提供、出售、转让、再许可给任何他人用于华为许可以外的任何其他用途。

Copyright ©【2025】【Huawei Technologies Co., Ltd.】. All Rights Reserved.

Without prior written permission of Huawei, no one is allowed to reproduce, modify, integrate, adapt, translate this Document, or provide, sell, transfer, sublicense this Document or any part of this Document or its derivative to any third party for any other purpose otherwise permitted by Huawei.

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息技术）全技术领域的认证体系，包含ICT基础设施、基础软硬件、云平台及云服务三类技术认证及项目管理认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

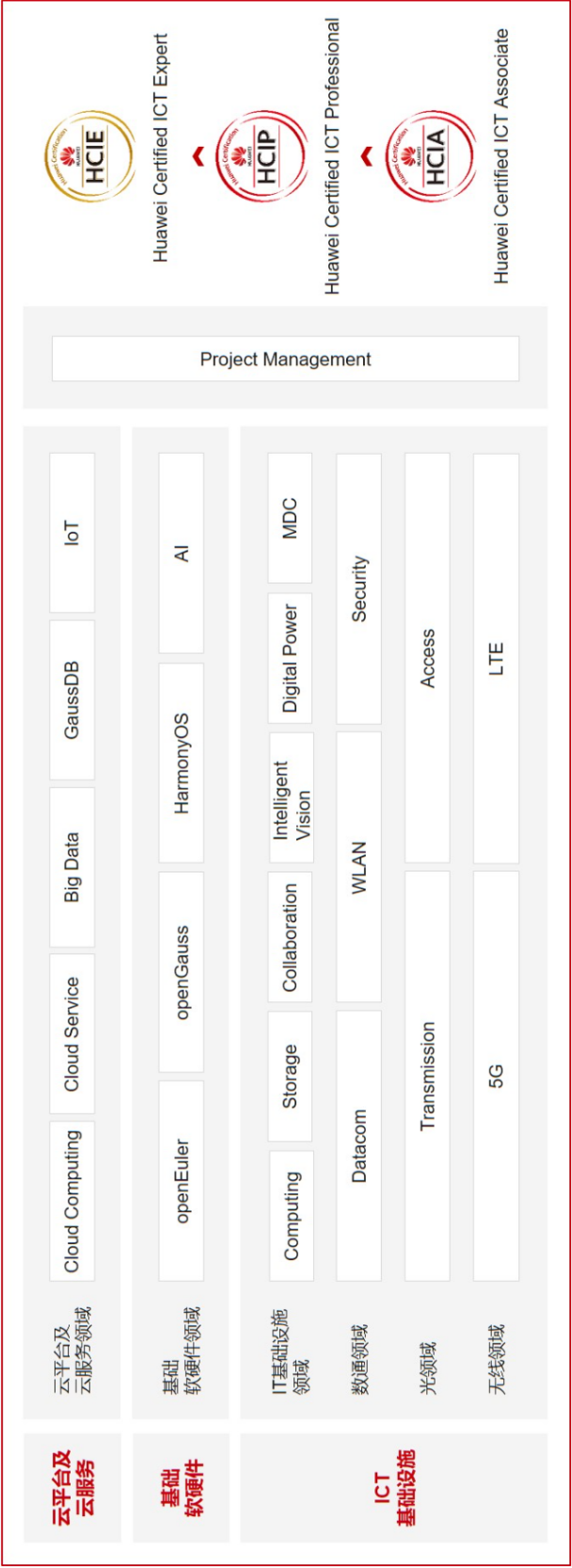
华为认证HCIP-AI-Model Developer V2.0定位于培养与认证具备模型设计与选型、算子开发、模型迁移、预训练与微调、模型评估与调优等模型开发全流程能力的高级工程师。

通过HCIP-AI-Model Developer V2.0认证，您将掌握AI计算系统基础知识和模型开发全流程知识，具备模型设计与选型、算子开发、模型迁移、预训练与微调和模型评估与调优等能力，胜任人工智能模型开发高级工程师、人工智能训练高级工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在AI智能世界的潮头浪尖！

华为职业认证全景图

华为职业认证覆盖ICT全领域，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。



简介

本书为 HCIP-AI-Model Developer 认证培训教程，适用于准备参加 HCIP-AI-Model Developer 考试的学员或者希望了解 PyTorch、数据工程、模型训练、算子开发、集合通信知识的读者。

内容描述

- 本实验指导书共包含 1 个实验，Qwen3-4B 微调实验。

实验环境说明

组网说明

本实验环境用于 HCIP-AI Model Developer V2.0 预训练实验。每套实验环境包括一台 windows 系统电脑，一个华为云 OBS 桶，一台 ModelArts 服务器。每套实验环境适用于 1 位学员操作。

设备介绍

为了满足 HCIP-AI Model Developer V2.0 实验需要，建议每套实验环境采用以下配置：

设备名称、型号与版本的对应关系如下：

设备名称	区域/设备型号
OBS桶	西南-贵阳一
obsutil	obsutil_windows64
ModelArts	Ascend: 1*ascend-snt9b1 ARM: 24核 192GB



目录

前 言 3

简介 3

内容描述 3

实验环境说明 3

1 Qwen3-4B 微调训练 5

1.1 实验介绍 5

1.1.1 关于本实验 5

1.2 实验设备介绍 5

1.3 实验准备 5

1.3.1 创建 ModelArts 实例 5

1.3.2 下载模型 8

1.3.3 安装依赖包 8

1.3.4 下载数据和处理数据 10

1.3.5 hf2mcore 权重转换 19

1.4 启动 LoRA 微调 20

1.5 思考题 25

1.6 实验总结 25

1

Qwen3-4B 微调训练

1.1 实验介绍

1.1.1 关于本实验

Qwen 是阿里巴巴集团 Qwen 团队研发的大语言模型和大型多模态模型系列。无论是语言模型还是多模态模型，均在大规模多语言和多模态数据上进行预训练，并通过高质量数据进行后期微调以贴近人类偏好。Qwen 具备自然语言理解、文本生成、视觉理解、音频理解、工具使用、角色扮演、作为 AI Agent 进行互动等多种能力。4B 是其中一个低参数量的版本模型。

1.2 实验设备介绍

设备型号与软件版本的对应关系如下，本实验中使用 ModelArts 提供的镜像。如果使用其他昇腾设备，环境相关适配软件请参见下表，昇腾适配软件请参照官网适配版本安装。

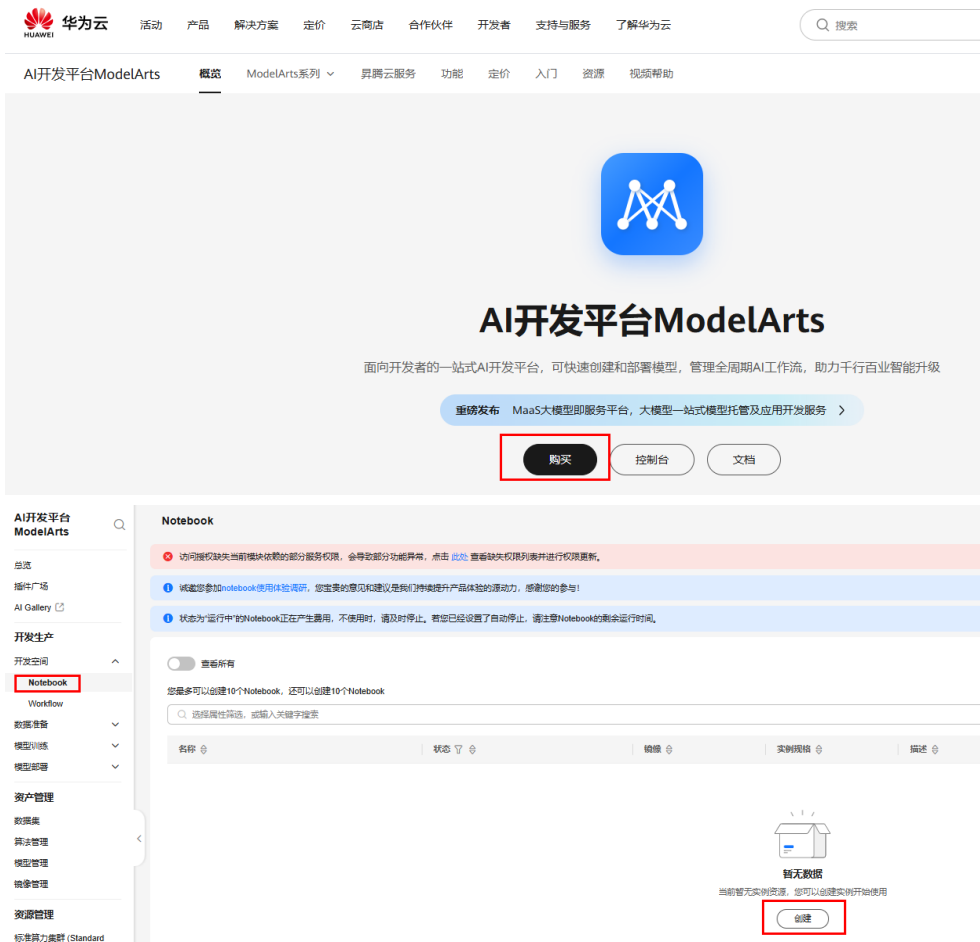
硬件设备型号	软件名称	软件版本
Atlas A2 系列训练设备	CANN	8.0.0
	Python	3.10
	PyTorch	2.1.0

1.3 实验准备

1.3.1 创建 ModelArts 实例

步骤 1 登陆 ModelArts: <https://www.huaweicloud.com/product/modelarts.html>。

步骤 2 购买 ModelArts 上 Notebook，点击购买，进入购买界面，选择 Notebook，点击创建。

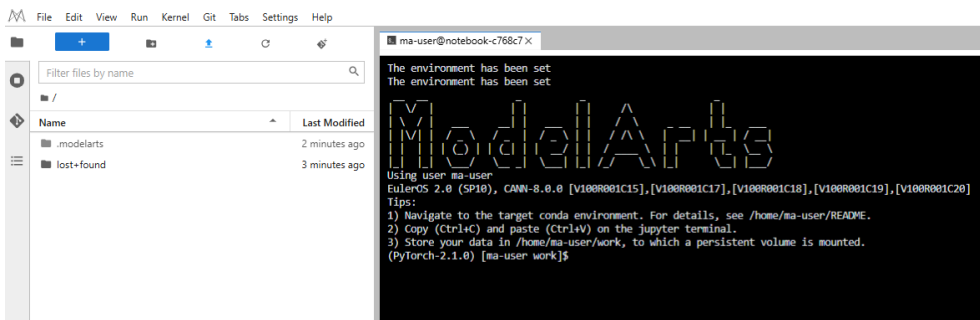


步骤 3 选择区域为“西南-贵阳一”，对 Notebook 命名，设置自动启停时间，按需设置时长。



步骤 4 镜像、服务器及硬盘大小按下图选择。

The screenshot shows the JupyterLab interface. On the left is a sidebar with a file browser. The main area is titled 'Launcher' and displays a grid of application icons. The 'Terminal' icon, which is a black square with a white '\$' symbol, is highlighted with a red rectangular box. Other icons include Notebook, PyTorch 2.1.0, python 3.10.6, Console, PyTorch 3.10.6, Other, VS Code, Performance Advisor, Text File, Markdown File, Pipeline As Code, TensorBoard, Python File, and Notebook Jobs.



1.3.2 下载模型

步骤 1 创建保存模型的文件夹。

```
mkdir qwen && cd qwen
```

步骤 2 安装依赖。

```
pip install -U huggingface_hub
```

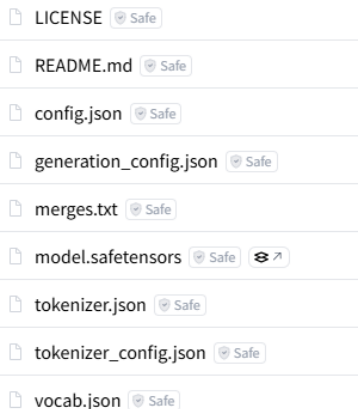
步骤 3 设置环境变量。

```
export HF_ENDPOINT=https://hf-mirror.com
```

步骤 4 下载模型。

```
cd ..
huggingface-cli download --resume-download Qwen/Qwen3-4B --local-dir qwen
```

下载完成模型后包含以下文件：



1.3.3 安装依赖包

1.3.3.1 安装 APEX

APEX 混合精度模块是一个集优化性能、精度收敛于一身的综合优化库，可以提供不同场景下的混合精度训练支持。

混合精度训练是在训练时混合使用单精度（float32）与半精度(float16)数据类型，将两者结合在一起，并使用相同的超参数实现了与 float32 几乎相同的精度。在迁移完成、训练开始之前，基于 NPU 硬件的架构特性，用户可以选择开启混合精度，用以提升模型的性能。

步骤 1 请确保已安装 PyTorch 框架且 setuptools 版本小于等于 65.7.0，若版本不符合条件，可使用以下命令安装。

通过 cd 命令回到进入 Notebook 时的目录（例如 ma-user/work），然后执行下面的命令。

```
pip3 install setuptools==65.7.0
```

步骤 2 获取昇腾适配的 APEX 源码。

```
git clone -b master https://gitee.com/ascend/apex.git
```

步骤 3 进入昇腾适配的 APEX 源码目录，执行命令编译生成二进制安装包。

```
cd apex && bash scripts/build.sh --python=3.10
```

步骤 4 执行如下命令进行安装。如果使用非 root 用户安装，需要在命令后加--user。

```
cd apex/dist/  
pip3 uninstall apex      # 可选，若当前未安装 apex，可不执行  
pip3 install apex-0.1+ascend-cp3x-cp3x-arch.whl    # x 指 Python 版本尾号，arch 指 CPU 架构
```

命令示例：

```
# 若用户在 x86 架构下安装，请将命令中文件包名中的“aarch64”改为“x86_64”。  
pip3 install apex-0.1+ascend-cp310-cp310-linux_aarch64.whl
```

1.3.3.2 安装 MindSpeed-LLM

步骤 1 初始化 Ascend 环境。

```
source /usr/local/Ascend/ascend-toolkit/set_env.sh
```

步骤 2 安装 MindSpeed-LLM 及相关依赖。

```
# 使能环境变量  
source /usr/local/Ascend/ascend-toolkit/set_env.sh  
source /usr/local/Ascend/nnal/atb/set_env.sh  
  
# 安装 MindSpeed 加速库  
git clone https://gitee.com/ascend/MindSpeed.git  
cd MindSpeed  
git checkout 2c085cc9 # checkout commit from MindSpeed core_r0.8.0 in 2025.04.01  
pip install -r requirements.txt  
pip3 install -e .  
cd ..  
  
# 准备 MindSpeed-LLM 及 Megatron-LM 源码
```

```
git clone https://gitee.com/ascend/MindSpeed-LLM.git
git clone https://github.com/NVIDIA/Megatron-LM.git # megatron 从 github 下载，请确保网络能访问
cd Megatron-LM
git checkout core_r0.8.0
cp -r megatron ../MindSpeed-LLM/
cd ../MindSpeed-LLM
git checkout 2.1.0

pip install -r requirements.txt # 安装其余依赖库
```

qwen3 系列模型依赖 transformers 4.51.0, 需要在环境配置完成后手动执行 `pip install transformers==4.51.0`。

```
pip install transformers==4.51.0
```

1.3.4 下载数据和处理数据

步骤 1 下载数据

本次使用 Alpaca 风格微调数据集，可以基于网页直接下载，首先回到 work 目录下。

```
# 创建文件夹保存数据
mkdir dataset
cd dataset/
# 下载数据
wget https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet
cd ..
```

步骤 2 处理数据

首先找到转换数据的 shell 文件，本次使用的是 Qwen3-4B 模型，

```
cd MindSpeed-LLM/
```

进入 shell 脚本，修改路径，

```
vim examples/mcore/qwen3/data_convert_qwen3_instruction.sh
```

修改 shell 脚本，

```
source /usr/local/Ascend/ascend-toolkit/set_env.sh # 初始化
mkdir ./finetune_dataset # 创建保存转换数据的路径，不需要修改

python ./preprocess_data.py \
    # 下面的命令是保存数据的路径，请# 找到自己保存路径的位置，复制绝对路径，替换./dataset/train-00000-of-00001-a09b74b3ef9c3b56.parquet
    --input ./dataset/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
    # 下面的命令中，请找到下载模型的文件夹路径，使用绝对路径替换./moeL_from_hf/qwen3_hf/
    --tokenizer-name-or-path ./moeL_from_hf/qwen3_hf/ \
    # 下面的命令中，替换为保存数据的绝对路径，注意，绝对路径后一定要加 alpaca
    --output-prefix ./finetune_dataset/alpaca \
```

```
--handler-name AlpacaStyleInstructionHandler \ # 不需要修改
--tokenizer-type PretrainedFromHF \ # 不需要修改
--workers 4 \ # 不需要修改
--log-interval 1000 \ # 不需要修改
--prompt-type qwen # 不需要修改
```

保存后执行脚本，

```
bash examples/mcore/qwen3/data_convert_qwen3_instruction.sh
```

```
INFO:mindspeed_llm.tasks.preprocess.templates:Replace eos token: <|lm_end|>
INFO:_main__building dataset: /home/ma-user/work/dataset/train-00000-of-00001-a09b7d3ef9c356.parquet
INFO:mindspeed_llm.tasks.preprocess.data_handler:Loading data from local file, format: parquet, file num: 1
Setting num_proc from 4 back to 1 for the train split to disable multiprocessing as it only contains one shard.
WARNING:datasets.builder:Setting num_proc from 4 back to 1 for the train split to disable multiprocessing as it only contains one shard.
Generating train split: 52002 examples (00:00, 101550.62 examples/s)
Converting format of dataset (num_proc=4): 100% |#####| 52002/52002 [00:03:00:00, 16380.30 examples/s]
```

等待数据转换完成。

```
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 34000 documents (3254.447063658401 docs/s, 3.4717216405682714 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 35000 documents (3252.266296884715 docs/s, 3.471786757183844 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 36000 documents (3250.5865729186203 docs/s, 3.473888939754259 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 37000 documents (3252.47640208314025 docs/s, 3.4745904488275436 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 38000 documents (3253.9227289583196 docs/s, 3.474244769075447 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 39000 documents (3254.397981204382 docs/s, 3.475805583751114 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 40000 documents (3255.1178633570453 docs/s, 3.4757883038448583 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 41000 documents (3257.372824886575 docs/s, 3.4763785512903476 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 42000 documents (3256.3257910174416 docs/s, 3.4780282611837694 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 43000 documents (3255.596155547811 docs/s, 3.4793842531197336 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 44000 documents (3254.8030906254353 docs/s, 3.4798116782729758 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 45000 documents (3263.006715449458 docs/s, 3.4874504535817645 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 46000 documents (3264.2185662652428 docs/s, 3.487241837439608 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 47000 documents (3263.434348584645 docs/s, 3.4872785354516302 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 48000 documents (3262.3373956677433 docs/s, 3.4883953308018186 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 49000 documents (3262.9077069888464 docs/s, 3.4880517155891385 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 50000 documents (3262.0144909969375 docs/s, 3.490158597082692 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 51000 documents (3262.229706767483 docs/s, 3.4907165551213635 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Processed 52000 documents (3260.0889096372343 docs/s, 3.4887196017338047 MB/s).
INFO:mindspeed_llm.tasks.preprocess.data_handler:Skip 0.0 sample exceeded seq-length(None)
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.10/tempfile.py:860: ResourceWarning: Implicitly cleaning up <TemporaryDirectory '/tmp/tmpjflndlnl'>
```

步骤 3 分析数据处理代码，以下针对重点代码进行分析注释。

```
try:
    import nltk
except ImportError:
    nltk = None # NLTK 用于句子分割，若未安装则设为 None

# 将上级目录添加到系统路径，以便导入自定义模块
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), os.path.pardir)))

# 导入自定义模块中的功能
from megatron.core.datasets.indexed_dataset import (
    IndexedDatasetBuilder, # 用于构建索引数据集
    IndexedDataset, # 索引数据集类
    get_bin_path, # 获取二进制文件路径
    get_idx_path, # 获取索引文件路径
)
from mindspeed_llm.training.tokenizer import build_tokenizer # 构建分词器函数
from mindspeed_llm.tasks.preprocess.data_handler import build_dataset, get_dataset_handler # 数据集构建相关函数

# 配置日志
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

```
# 获取当前文件目录
cur_file_dir = Path(__file__).absolute().parent
# 模板文件路径，用于微调时的提示模板
TEMPLATES_DIR = os.path.join(cur_file_dir, "configs/finetune/templates.json")

# 自定义语言变量类，用于 NLTK 句子分割时保留换行符
class CustomLanguageVars(nltk.tokenize.punkt.PunktLanguageVars if nltk else object):
    # 定义句子结束符的上下文格式，修改了默认的空格处理方式以保留换行符
    _period_context_fmt = r"""
        \S*                                # 一些单词材料
        %(SentEndChars)s                    # 潜在的句子结束符
        \S*                                # 调整了空格处理（原代码中的修改点）
        (?=(?P<after_tok>
            %(NonWord)s                      # 其他标点符号
            |
            (?P<next_tok>\S+)                # 下一个单词（通常这里应该是\S+）
        ))"""

# 身份分割器，不进行实际分割，直接返回输入文本
class IdentitySplitter(object):
    def tokenize(self, *text):
        return text # 直接返回输入文本，不做处理

# 构建文本分割器函数
def build_splitter(args):
    if nltk and args.split_sentences:
        nltk.download("punkt", quiet=True) # 下载 NLTK 的 punkt 分词模型
    if args.split_sentences:
        if not nltk:
            logger.error("NLTK is not available to split sentences.")
            raise Exception("nltk is not available")
        # 加载英文句子分割模型
        splitter = nltk.load("tokenizers/punkt/english.pickle")
        if args.keep_newlines:
            # 自定义分割器，防止 punkt 吃掉句子后的换行符
            final_splitter = nltk.tokenize.punkt.PunktSentenceTokenizer(
                train_text=splitter._params,
                lang_vars=CustomLanguageVars())
        else:
            final_splitter = splitter
    else:
        final_splitter = IdentitySplitter() # 不分割时使用身份分割器
    return final_splitter
```

```
# 添加数据相关参数到命令行解析器
def add_data_args(parser):
    group = parser.add_argument_group(title='input data')
    group.add_argument('--input', type=str, required=True,
                        help='输入 JSON 路径、目录路径或 Hugging Face 数据集名称；合并数据集时为包含所有文档文件的目录路径')
    group.add_argument('--handler-name', type=str, default="",
                        help='指定数据集处理器')
    group.add_argument('--streaming', action='store_true',
                        help='是否使用流式处理')
    group.add_argument('--hf-datasets-params', default=None,
                        help='Hugging Face 加载数据集的参数')
    group.add_argument('--datasets', nargs='+', default=None,
                        help='要合并的一个或多个输入数据集路径')
    group.add_argument('--json-keys', nargs='+', default=['text'],
                        help='从 JSON 中提取的键列表')
    group.add_argument('--split-sentences', action='store_true',
                        help='将文档分割成句子')
    group.add_argument('--keep-newlines', action='store_true',
                        help='分割时保留句子之间的换行符')
    # LlamaFactory 相关参数，用于构建训练提示
    group.add_argument('--prompt-type', type=str, default=None,
                        choices=['default', 'empty', 'trl', 'chatglm2', 'chatglm3', 'chatglm3_system',
                                'glm4', 'chatml',
                                'chatml_de', 'qwen', 'qwen_r1', "qwen_math_r1", 'llama3', 'llama2',
                                'mistral', 'mixtral', 'gemma', 'alpaca',
                                'deepseek2', 'deepseek2-lite', 'cpm', 'baichuan2', 'deepseek3', 'intern2',
                                'hunyuan'],
                        help='训练中构建提示使用的模板类型，如"qwen"')
    group.add_argument('--prompt-type-path', type=str, default=TEMPLATES_DIR,
                        help='提示模板 JSON 文件路径')
    group.add_argument('--dataset-additional-keys',
                        nargs='*',
                        default=[],
                        help='需要从数据集中添加的额外键')
    group.add_argument("--interleave-probs", default=None,
                        help='从数据集中采样数据的概率，用逗号分隔，概率和应为 1，如"0.1, 0.2, 0.3, 0.4"')
    group.add_argument('--mix-strategy', type=str,
                        default='concat',
                        choices=['concat', 'interleave_under', 'interleave_over'],
                        help='数据集混合策略（连接/交织）（欠采样/过采样）')
    group.add_argument("--dataset-dir", default=None,
                        help="包含数据集的文件夹路径")
    group.add_argument("--overwrite-cache", action='store_true',
```

```
        help="覆盖缓存的训练和评估集")
group.add_argument("--max-samples", type=int, default=None,
                    help="调用，截断每个数据集的样本数")
group.add_argument("--seed", type=int, default=1234,
                    help="数据混合使用的随机种子")
group.add_argument("--cache-dir", type=str, default="~/tmp",
                    help="本地数据集缓存位置")
group.add_argument("--map-keys", type=json.loads, default=None,
                    help="数据集字段映射")
group.add_argument("--pack", action='store_true',
                    help="在微调数据集中将多个样本打包成一个样本")
group.add_argument("--neat-pack", action='store_true',
                    help="使用锯齿形注意力掩码")
group.add_argument("--script-data-dir", type=str, default=None,
                    help="Python 脚本数据集方向")
group.add_argument("--pad-to-multiple-of", type=int, default=1,
                    help="将每个数据填充到指定倍数")

# 添加分词器相关参数到命令行解析器
def add_tokenizer_args(parser):
    group = parser.add_argument_group(title='tokenizer')
    group.add_argument('--tokenizer-type', type=str, default='PretrainedFromHF',
                        choices=['BertWordPieceLowerCase', 'BertWordPieceCase',
                                'GPT2BPETokenizer', 'GPTSentencePieceTokenizer',
                                'PretrainedFromHF'],
                        help='使用的分词器类型')
    group.add_argument("--tokenizer-not-use-fast", action='store_false',
                        help="HuggingFace 分词器不使用快速版本")
    group.add_argument('--vocab-file', type=str, default=None,
                        help='词汇表文件路径')
    group.add_argument('--merge-file', type=str, default=None,
                        help='BPE 合并文件路径（如有需要）')
    group.add_argument('--append-eod', action='store_true',
                        help='在文档末尾添加<eod>标记')
    group.add_argument("--tokenizer-name-or-path", type=str, default=None,
                        help="HuggingFace 分词器的名称或路径")
    group.add_argument("--tokenizer-model", type=str, default=None,
                        help="分词器模型文件")
    group.add_argument('--seq-length', type=int, default=None,
                        help='处理的最大序列长度')
    group.add_argument('--make-vocab-size-divisible-by', type=int, default=128,
                        help='将词汇表大小填充为该值的倍数，用于计算效率')
    group.add_argument('--pad-vocab-size-to', type=int, default=None,
                        help='将词汇表大小填充到指定值，该值必须大于初始大小，使用此参数时将忽略`make-vocab-size-divisible-by`')
    group.add_argument(
```

```
--placeholder-token',
default='κ',
help="标记 PRM 可进行预测的每个步骤结束的特殊占位符标记")
group.add_argument(
    '--reward-tokens',
    nargs='+',
    type=str,
    default=[],
    help="表示整个推理过程中每个推理步骤正确性的标签")
parser.add_argument('--trust-remote-code',
                    action='store_true',
                    default=False,
                    help='启用信任远程代码以加载 transformer 模型')

# 添加输出相关参数到命令行解析器
def add_output_args(parser):
    group = parser.add_argument_group(title='output data')
    group.add_argument('--output-prefix', type=str, required=True,
                      help='不带后缀的二进制输出文件路径前缀')
    group.add_argument('--dataset-impl', type=str, default='mmap',
                      choices=['lazy', 'cached', 'mmap'])

    group = parser.add_argument_group(title='runtime')
    group.add_argument('--workers', type=int, default=1,
                      help='启动的工作进程数')
    group.add_argument('--n-subs', type=int, default=1,
                      help='多进程处理时切割的子集数')
    group.add_argument('--log-interval', type=int, default=100,
                      help='进度更新间隔')

# 添加合并数据集相关参数到命令行解析器
def add_merge_args(parser):
    group = parser.add_argument_group(title='merge data')
    group.add_argument('--merge-group-keys', nargs='+', default=None, const=None,
                      help='文件名中具有相同 key 的`bin-idx`对文件将被合并')

# 获取命令行参数
def get_args():
    parser = argparse.ArgumentParser()

    add_data_args(parser)
    add_tokenizer_args(parser)
    add_output_args(parser)
```

```
add_merge_args(parser)

args = parser.parse_args()
args.keep_empty = False # 不保留空数据

if args.tokenizer_type.lower().startswith('bert'):
    if not args.split_sentences:
        logger.warning("检测到 Bert 分词器，确定不分割句子吗？")

# 为分词器设置一些默认/虚拟值
args.rank = 0
args.tensor_model_parallel_size = 1
args.vocab_extra_ids = 0

return args

# 验证参数有效性
def validate_args(args):
    support_prompt_type_handler = [
        "LlamaFactoryInstructionHandler",
        "AlpacaStyleInstructionHandler",
        "SharegptStyleInstructionHandler",
        "AlpacaStylePairwiseHandler",
        "SharegptStylePairwiseHandler",
        "AlpacaStyleProcessRewardHandler",
        "PPOAlpacaStyleInstructionHandler",
        "HunyuanInstructionHandler",
        "R1AlpacaStyleInstructionHandler",
        "R1SharegptStyleInstructionHandler"
    ]
    # 验证提示类型和处理器名称的匹配性
    if args.prompt_type is not None and args.handler_name not in support_prompt_type_handler:
        raise AssertionError(f'如果指定 prompt_type，处理器名称必须在以下列表中:\n{support_prompt_type_handler}.')

    if (args.merge_group_keys is not None) and (not os.path.isdir(args.input)):
        raise ValueError(f'{args.input} 不是目录或不存在')

    if not os.path.isdir(os.path.dirname(args.output_prefix)):
        raise ValueError(f'{os.path.dirname(args.output_prefix)} 不是目录或不存在')

    if not args.pack and args.neat_pack:
        raise ValueError("设置`--neat-pack`时需要设置`--pack`")

# 将范围切割为多个子集
```

```
def cut_range_to_subs(n, gap):
    n_ = n // gap
    mod = n % gap
    if mod != 0:
        return [(k * gap, (k + 1) * gap) for k in range(0, n_)] + [(gap * n_, n)]
    else:
        return [(k * gap, (k + 1) * gap) for k in range(0, n_)]

# 处理数据子集的函数（用于多进程）
def handle_subset(params):
    """处理数据子集并序列化为磁盘上的索引数据集"""
    handler = get_dataset_handler(params[0], params[1], params[2], params[3])
    handler.serialize_to_disk() # 序列化到磁盘
    return handler.output_idx_files # 返回输出的索引文件路径

# 合并数据集函数
def merge_datasets(args):
    # 按 merge_group_keys 分组管理前缀
    prefixes = {key: set() for key in args.merge_group_keys}
    for key in prefixes:
        for basename in os.listdir(args.input):
            prefix, ext = os.path.splitext(basename)

            if prefix in prefixes[key] or key not in prefix:
                continue

            if not os.path.isfile(os.path.join(args.input, basename)):
                continue

            # 检查对应的 bin/idx 文件是否存在
            ext_pair = ".bin" if ext == ".idx" else ".idx"
            if not os.path.isfile(os.path.join(args.input, prefix) + ext_pair):
                raise FileNotFoundError(f"未提供{os.path.join(args.input, prefix)}的{ext_pair}文件")

            prefixes[key].add(prefix)

# 合并每个 key 对应的数据集
for key in prefixes:
    builder = None
    for prefix in sorted(prefixes[key]):
        if builder is None:
            # 加载第一个数据集以获取数据类型
            dataset = IndexedDataset(os.path.join(args.input, prefix), multimodal=False)
            builder = IndexedDatasetBuilder(
```

```
        get_bin_path(f'{args.output_prefix}_{key}'), dtype=dataset.index.dtype,
        multimodal=False
    )
    del dataset  # 释放内存

    # 将当前前缀的索引添加到构建器
    builder.add_index(os.path.join(args.input, prefix))

    # 完成合并并生成最终的索引文件
    builder.finalize(get_idx_path(f'{args.output_prefix}_{key}'))

# 主函数
def main():
    args = get_args()  # 获取命令行参数
    validate_args(args)  # 验证参数有效性

    if args.merge_group_keys is not None:
        merge_datasets(args)  # 处理数据集合并
        return

    # 构建分词器
    tokenizer = build_tokenizer(args)
    # 构建文本分割器
    splitter = build_splitter(args)

    logger.info("building dataset: %s", args.input)
    # 构建原始数据集
    raw_data = build_dataset(args)

    if args.n_subs == 1:
        # 单进程处理
        handler = get_dataset_handler(args, raw_data, tokenizer, splitter)
        # 序列化到 bin 和 idx 文件
        handler.serialize_to_disk()
    else:
        # 多进程处理
        target_prefix = args.output_prefix
        target_prefixname = os.path.basename(target_prefix)

        num_samples = len(raw_data)
        # 将数据集切割为多个子集
        start_ends = cut_range_to_subs(num_samples, num_samples // args.n_subs)
        subsets = [raw_data.select(range(x[0], x[1])) for x in start_ends]

        # 准备多进程参数
```

```

params_list = []
for k, subset in enumerate(subsets):
    args_ = copy.deepcopy(args)
    # 为每个子集生成唯一的输出前缀
    args_.output_prefix = target_prefix.replace(target_prefixname,
f'{str(k).zfill(3)}_of_{str(len(subsets)-1).zfill(3)}_{target_prefixname}')
    params = [args_, subset, tokenizer, splitter]
    params_list.append(params)

# 创建多进程池并处理子集
pool = multiprocessing.Pool()
sub_idx_files = pool.map(handle_subset, params_list)
pool.close()
pool.join()

# 合并多进程处理结果
for key in sub_idx_files[0].keys():
    idx_files = [x[key] for x in sub_idx_files]
    idx_files.sort()
    # 生成最终的索引文件路径
    target_idx = idx_files[0].replace(f'000_of_{str(len(subsets)-1).zfill(3)}_{target_prefixname}',
target_prefixname)
    target_bin = target_idx.replace('.idx', '.bin')
    # 构建最终的索引数据集
    idx = IndexedDatasetBuilder(target_bin)
    for idx_file in idx_files:
        idx.add_index(idx_file.replace('.idx', ''))
    idx.finalize(target_idx) # 完成索引构建

# 清理临时文件
for idx_file in idx_files:
    os.remove(idx_file)
    os.remove(idx_file.replace('.idx', '.bin'))

```

1.3.5 hf2mcore 权重转换

这一步是需要把 hf 的模型权重转换为

步骤 1 修改转换脚本，进入道 qwen3 路径下，找到 ckpt_convert_qwen3_hf2mcore.sh。

```
vim examples/mcore/qwen3/ckpt_convert_qwen3_hf2mcore.sh
```

修改注释部分的脚本，

```

export CUDA_DEVICE_MAX_CONNECTIONS=1
source /usr/local/Ascend/ascend-toolkit/set_env.sh

python convert_ckpt.py \
    --use-mcore-models \

```

```
--model-type GPT \
--load-model-type hf \
--save-model-type mg \
--target-tensor-parallel-size 1 \
--target-pipeline-parallel-size 1 \    # 此处注意，如果是 8 卡修改成 8，单卡修改为 1
--spec mindspeed_llm.tasks.models.spec.qwen3_spec layer_spec \
# 修改下面的路径为模型的绝对路径
--load-dir ./model_from_hf/qwen3_hf/ \
# 修改为需要把转换的模型保存的绝对路径，不需要提前创建，保持当前路径也可
--save-dir ./model_weights/qwen3_mcore/ \
# 修改为模型中 tokenizer.json 的绝对路径
--tokenizer-model ./model_from_hf/qwen3_hf/tokenizer.json \
--params-dtype bf16 \
--model-type-hf qwen3
```

保存脚本，然后启动转换脚本，

```
bash examples/mcore/qwen3/ckpt_convert_qwen3_hf2mcore.sh
```

```
building GPT model ...
WARNING:mindspeed_llm.core.models.common.language_module.language_module:Distributed processes aren't initialized, so the output layer is not initialized with weights from the word embeddings. If you ar
e just manipulating a model this is fine, but this needs to be handled manually. If you are training something is definitely wrong.
[loading checkpoint shards: 0/1] | 0/3 [00:00<, 2117.8]

building GPT model ...
INFO:root:received transformer layer 4
INFO:root:received transformer layer 5
INFO:root:received transformer layer 6
INFO:root:received transformer layer 7
saving checkpoint at iteration 1 to ./model_weights/qwen3_mcore/ in torch format
successfully saved checkpoint from iteration 1 to ./model_weights/qwen3_mcore/
building GPT model ...
INFO:root:received transformer layer 8
INFO:root:received transformer layer 9
INFO:root:received transformer layer 10
INFO:root:received transformer layer 11
saving checkpoint at iteration 1 to ./model_weights/qwen3_mcore/ in torch format
successfully saved checkpoint from iteration 1 to ./model_weights/qwen3_mcore/
building GPT model ...
INFO:root:received transformer layer 12
INFO:root:received transformer layer 13
INFO:root:received transformer layer 14
INFO:root:received transformer layer 15
saving checkpoint at iteration 1 to ./model_weights/qwen3_mcore/ in torch format
```

等待转换完成。

1.4 启动 LoRA 微调

步骤 1 准备启动 LoRA 微调，需要修改启动 shell 脚本。

```
vim examples/mcore/qwen3/tune_qwen3_4b_4K_lora_ptd.sh
```

进入脚本，

```
vim tune_qwen3_0point6b_4K_lora_ptd.sh
```

修改脚本，下面注释部分是需要修改的部分，其余部分不需要修改。

```
export CUDA_DEVICE_MAX_CONNECTIONS=1
# Change for multinode config
```

```
NPUS_PER_NODE=1 # 单卡 1, 8 卡改为 8
MASTER_ADDR=localhost
MASTER_PORT=6015
NNODES=1
NODE_RANK=0
WORLD_SIZE=$((NPUS_PER_NODE*NNODES))

# please fill these path configurations
CKPT_LOAD_DIR="your model ckpt path" # 引号内修改为转换后模型的绝对路径
CKPT_SAVE_DIR="your model save ckpt path" # 引号内修改为保存 checkpoint 的绝对路径, 自动创建
DATA_PATH="your data path" # 引号内修改为转换后数据的绝对路径/ alpaca ( alpaca 加在路径后 )
TOKENIZER_PATH="your tokenizer path" # 引号内改为原始模型的路径

TP=1
PP=1
MBS=1
GBS=16
SEQ_LENGTH=8192
TRAIN_ITERS=2000
```

步骤 2 保存 shell 脚本, 然后执行。

```
bash examples/mcore/qwen3/tune_qwen3_4b_4K_lora_ptd.sh
```

打印基本信息,

```
when context_parallel is not activated, kv_head_repeat_before_uly_alltoall would be set to False for reducing memory usage.
[INFO] Setting args.create_attention_mask_in_data_loader to False since reset_data=False or alibi_without_flash_attn=False or args.tokenizer_padding_side=right
----- MindSpeed-LUM Arguments -----
accumulate_allreduce_grads_in_fp32 ..... True
adam_beta1 ..... 0.9
adam_beta2 ..... 0.95
adam_eps ..... 1e-08
adaptive_cp_dynamic_attn_mask ..... False
adaptive_cp_manually_set_mask_list ..... False
adaptive_cp_only_reschedule ..... False
adaptive_cp_without_coarse ..... False
adaptive_recompute_device_size ..... -1
adaptive_recompute_device_swap ..... False
adaptive_recompute_profiling_step ..... 10
add_bias_linear ..... False
add_dense_bias ..... False
add_eos_token ..... []
add_output_layer_bias ..... False
add_position_embedding ..... True
```

启动微调,

```
Resetting memory footprints: weight and optimizer=0.000128
Rank 0 (after 1 iterations) memory (MB) | allocated: 8168.98895783125 | max allocated: 9571.98771484375 | reserved: 9856.0 | max reserved: 9856.0
[2025-08-28 12:34:11] iteration 2/ 2000 | consumed samples: 32 | elapsed time per iteration (ms): 3637.4 | learning rate: 1.250000E-06 | global batch size: 16 | ln loss: 6.186918
v00 | loss scale: 1.0 | grad norm: 8.880 | number of skipped iterations: 0 | number of nan iterations: 0 |
[2025-08-28 12:34:11] iteration 3/ 2000 | consumed samples: 48 | elapsed time per iteration (ms): 3497.2 | learning rate: 1.675000E-06 | global batch size: 16 | ln loss: 4.558736
v00 | loss scale: 1.0 | grad norm: 6.931 | number of skipped iterations: 0 | number of nan iterations: 0 |
```

等待微调启动, 并结束。

步骤 3 分析启动脚本

启动 shell 脚本。

```
export HCCL_CONNECT_TIMEOUT=1800
export CUDA_DEVICE_MAX_CONNECTIONS=1

# 分布式训练基础配置
NPUS_PER_NODE=8 # 每个节点的 NPU 数量 ( 昇腾芯片数量 )
MASTER_ADDR=localhost # 主节点 IP ( 单节点训练时为 localhost )
MASTER_PORT=6014 # 主节点通信端口
```

```
NNODES=1          # 节点总数（单节点训练）
NODE_RANK=0        # 当前节点的排名（单节点为 0）
WORLD_SIZE=$((($NPUS_PER_NODE*$NNODES)) # 总进程数（8*1=8，对应 8 个 NPU，本次为 1*1）

# 路径配置
TP=1               # 张量并行（Tensor Parallel）大小（1 表示不使用张量并行）
PP=1               # 流水线并行（Pipeline Parallel）大小（1 表示不使用流水线并行）
SEQ_LENGTH=4096    # 最大序列长度（Qwen3-4B 支持的最大长度为 4096）
TRAIN_ITERS=2000   # 总训练迭代次数

# 模型并行与训练基础参数
TP=1               # 张量并行（Tensor Parallel）大小（1 表示不使用张量并行）
PP=1               # 流水线并行（Pipeline Parallel）大小（1 表示不使用流水线并行）
SEQ_LENGTH=4096    # 最大序列长度（Qwen3-4B 支持的最大长度为 4096）
TRAIN_ITERS=2000   # 总训练迭代次数

# 上面代码需要注意点：由于 Qwen3-4B 参数量较小（4B），这里未使用模型并行（TP=1, PP=1），
# 仅通过数据并行（8 个 NPU 分摊数据）加速训练。
# SEQ_LENGTH=4096 需与模型预训练时的序列长度保持一致，避免位置编码不匹配。

# 分布式训练参数
DISTRIBUTED_ARGS=""
    --nproc_per_node $NPUS_PER_NODE \ # 每个节点的进程数（与 NPU 数量一致）
    --nnodes $NNODES \                # 节点总数
    --node_rank $NODE_RANK \          # 当前节点排名
    --master_addr $MASTER_ADDR \      # 主节点地址
    --master_port $MASTER_PORT       # 主节点端口
"

# 优化器与算子融合参数
OPTIMIZE_ARGS=""
    --use-flash-attn \                # 使用 Flash Attention 加速注意力计算（高效内存利用）
    --use-fused-rotary-pos-emb \       # 使用融合算子计算旋转位置编码（RoPE）
    --use-rotary-position-embeddings \ # 启用旋转位置编码（Qwen 默认使用 RoPE）
    --use-fused-swiglu \              # 使用融合算子计算 Swiglu 激活函数
    --use-fused-rmsnorm \             # 使用融合算子计算 RMSNorm 归一化
    --no-masked-softmax-fusion \      # 不融合带掩码的 Softmax（视硬件兼容性调整）
    --use-distributed-optimizer \     # 使用分布式优化器（降低内存占用）
"

# 训练超参数
TRAIN_ARGS=""
    --micro-batch-size 1 \            # 每个进程的微批次大小（单 NPU 单次处理 1 个样本）
    --global-batch-size 16 \          # 全局批次大小（8 个 NPU * 1 * 2 梯度累积 = 16）
    --lr 1.25e-5 \                   # 学习率（LoRA 微调通常使用较小的学习率）
```

```
--lr-decay-style cosine \      # 学习率衰减方式（余弦衰减）
--min-lr 1.25e-7 \           # 最小学习率（衰减下限）
--weight-decay 1e-1 \        # 权重衰减（正则化，LoRA 微调可适当增大）
--lr-warmup-fraction 0.01 \   # 学习率热身比例（1%的迭代次数用于热身）
--attention-dropout 0.0 \     # 注意力 dropout（微调时通常关闭以避免欠拟合）
--hidden-dropout 0.0 \       # 隐藏层 dropout（同上）
--clip-grad 1.0 \            # 梯度裁剪阈值（防止梯度爆炸）
--adam-beta1 0.9 \           # Adam 优化器 beta1 参数
--adam-beta2 0.95 \          # Adam 优化器 beta2 参数
--initial-loss-scale 4096 \   # 混合精度训练的初始损失缩放
--seed 42 \                  # 随机种子（保证实验可复现）
--bf16 \                     # 使用 BF16 混合精度训练（昇腾支持良好）
--train-iters ${TRAIN_ITERS} \ # 总训练迭代次数（2000）
--seq-length ${SEQ_LENGTH} \  # 序列长度（4096）
--no-shared-storage \        # 不使用共享存储（单节点训练无需共享）
"

# 模型并行参数，本实验均为 1，单卡微调
MODEL_PARALLEL_ARGS="
    --tensor-model-parallel-size ${TP} \ # 张量并行大小（1）
    --pipeline-model-parallel-size ${PP} \ # 流水线并行大小（1）
"

# 模型结构参数
GPT_ARGS="
    --use-mcore-models \      # 使用 Megatron-Core 的模型结构
    --spec mindspeed_llm.tasks.models.spec.qwen3_spec layer_spec \ # Qwen3 的模型配置文件
    --kv-channels 128 \       # K/V 向量的通道数（隐藏层大小/注意力头数=2560/32=80？
需与模型匹配）
    --qk-layernorm \          # 对 Q/K 进行 LayerNorm（Qwen 的特性）
    --tokenizer-name-or-path ${TOKENIZER_PATH} \ # 分词器路径
    --max-position-embeddings ${SEQ_LENGTH} \   # 最大位置嵌入（4096）
    --num-layers 36 \          # 模型层数（Qwen3-4B 为 36 层）
    --hidden-size 2560 \       # 隐藏层大小（2560，对应 4B 参数量）
    --ffn-hidden-size 9728 \   # FFN 中间层大小（9728）
    --num-attention-heads 32 \  # 注意力头数（32）
    --tokenizer-type PretrainedFromHF \ # 分词器类型（Hugging Face 预训练分词器）
    --make-vocab-size-divisible-by 1 \ # 词汇表大小无需填充（Qwen 的词汇表已处理）
    --padded-vocab-size 151936 \ # 词汇表大小（Qwen 的 tokenizer 词汇量）
    --rotary-base 1000000 \     # RoPE 的基底（1e6，Qwen 的特性）
    --disable-bias-linear \     # 禁用线性层偏置（Qwen 无偏置设计）
    --position-embedding-type rope \ # 位置嵌入类型（RoPE）
    --normalization RMSNorm \   # 归一化方式（RMSNorm）
    --swiglu \                  # 激活函数（Swiglu）
```

```

--attention-softmax-in-fp32 \      # 注意力 Softmax 在 FP32 中计算（保证精度）
--no-gradient-accumulation-fusion \ # 不融合梯度累积（视硬件调整）
--group-query-attention \         # 启用分组查询注意力（GQA, Qwen3 的特性）
--num-query-groups 8 \           # 查询组数量（8 组，32 头/8 组=4 头/组）
"

# 数据参数
DATA_ARGS="
--data-path $DATA_PATH \      # 训练数据路径（预处理后的 bin/idx 文件）
--split 100,0,0 \            # 训练/验证/测试集比例（100%用于训练，无验证/测试）
"

# 输出与日志参数
OUTPUT_ARGS="
--load ${CKPT_LOAD_DIR} \     # 加载预训练模型权重
--save ${CKPT_SAVE_DIR} \     # 保存微调后模型
--log-interval 1 \            # 每 1 次迭代输出日志
--save-interval ${TRAIN_ITERS} \ # 训练结束后保存模型（2000 次迭代）
--eval-interval ${TRAIN_ITERS} \ # 训练结束后评估（这里 eval-iters=0 表示不评估）
--eval-iters 0 \              # 不进行评估
--no-load-optim \             # 不加载预训练的优化器状态（从头训练优化器）
--no-load-rng \               # 不加载预训练的随机数状态（重新初始化随机数）
"

# LoRA 微调参数
TUNE_ARGS="
--finetune \                  # 启用微调模式（而非预训练）
--stage sft \                 # 微调阶段：监督微调（Supervised Fine-Tuning）
--is-instruction-dataset \    # 标记数据集为指令数据集（需按指令格式处理）
--tokenizer-not-use-fast \    # 不使用 Hugging Face 的快速分词器（兼容性调整）
--prompt-type qwen \          # 使用 Qwen 的提示模板（如<|im_start|>user 等格式）
--variable-seq-lengths \      # 支持可变长度序列（不强制 padding 到固定长度）
--lora-r 16 \                  # LoRA 的秩（秩越小，参数量越少，默认 16）
--lora-alpha 32 \             # LoRA 的缩放因子（alpha=32，通常设为 2*r）
--lora-fusion \               # 启用 LoRA 融合（可能指训练后合并 LoRA 权重）
--lora-target-modules linear_qkv linear_proj linear_fc1 linear_fc2 # LoRA 目标模块
"

# 上面代码中，--lora-r 16 和--lora-alpha 32 控制 LoRA 的低秩矩阵大小
# --lora-target-modules 指定对哪些模块应用 LoRA：注意力的 QKV 线性层（linear_qkv）、
# 输出投影层（linear_proj）、前馈网络的两层（linear_fc1/2），这些是微调中最关键的模块
# --prompt-type qwen 确保输入按 Qwen 的指令格式（如用户 / 助手区分）处理，
# 避免格式不匹配导致性能下降。

# 启动命令
torchrun $DISTRIBUTED_ARGS posttrain_gpt.py \

```

```
$GPT_ARGS \  
$DATA_ARGS \  
$OUTPUT_ARGS \  
$OPTIMIZE_ARGS \  
$TRAIN_ARGS \  
$TUNE_ARGS \  
$MODEL_PARALLEL_ARGS \  
--distributed-backend nccl \ # 分布式后端（昇腾上实际使用 HCCL，此处兼容设置）  
| tee logs/tune_qwen3_4b_lora_ptd.log # 日志输出到文件
```

1.5 思考题

请各位同学思考，如何进行模型推理呢？

思路：使用 transformers 库加载模型，然后进行推理。同时 MindSpeed-LLM 提供了推理代码，大家可以分析学习。

详细推理请继续学习《HCIP-AI-Application Developer》相关内容，进行模型部署推理。

1.6 实验总结

本实验使用了 Qwen3-4B 进行了微调，实验包含了环境配置、模型权重下载、数据集下载、训练代码分析等。
