昇思学习打卡营·NLP特辑 第六课

玩转香橙派开发板

——基于MindSpore的AI开发实战

(模型训练+LLM推理应用)

陆铭皓

昇思 MindSpore

香橙派简介

OrangePi AIpro(8-12T)采用昇腾AI技术路线，具体为4核64位处理器+AI处理器，集成图形处理器，支持8-12TOPS AI算力，拥有8GB/16GB LPDDR4X，可以外接32GB/64GB/128GB/256GB eMMC模块，支持双4K高清输出。

丰富的接口：包括两个HDMI输出、GPIO接口、Type-C电源接口、支持SATA/NVMe SSD 2280的M.2插槽、TF插槽、千兆网口、两个USB3.0、一个USB Type-C 3.0、一个Micro USB（串口打印调试功能）、两个MIPI摄像头、一个MIPI屏等，预留电池接口。

广泛适的应用场景：可用于AI边缘计算、深度视觉学习及视频流AI分析、视频图像分析、自然语言处理、智能小车、机械臂、人工智能、无人机、云计算、AR/VR、智能安防、智能家居等领域，覆盖 AIoT各个行业。

系统支持：Orange Pi AIpro支持Ubuntu、openEuler操作系统，满足大多数AI算法原型验证、推理应用开发的需求。

Mindspore与香橙派

昇思MindSpore——香橙派上唯一支持训推的深度学习框架。
支持CV、NLP等小模型，满足规格不超过3B的大模型带框架推理。

实现动静统一，动态图+类HuggingFace的编程方式——灵活友好的模型开发与调试体验。

静态图获得更优的执行性能，达成推理性能的提升。
仅需两行代码即可实现动静态图的转换。

环境搭建

在使用香橙派之前，除了官方提供了一些配件外，还有一些需要自行提前准备的东西，自行提供的配件清单：
TF读卡器 HDMI显示器连接线 串口测试线

官方提供配件

自行提供配件

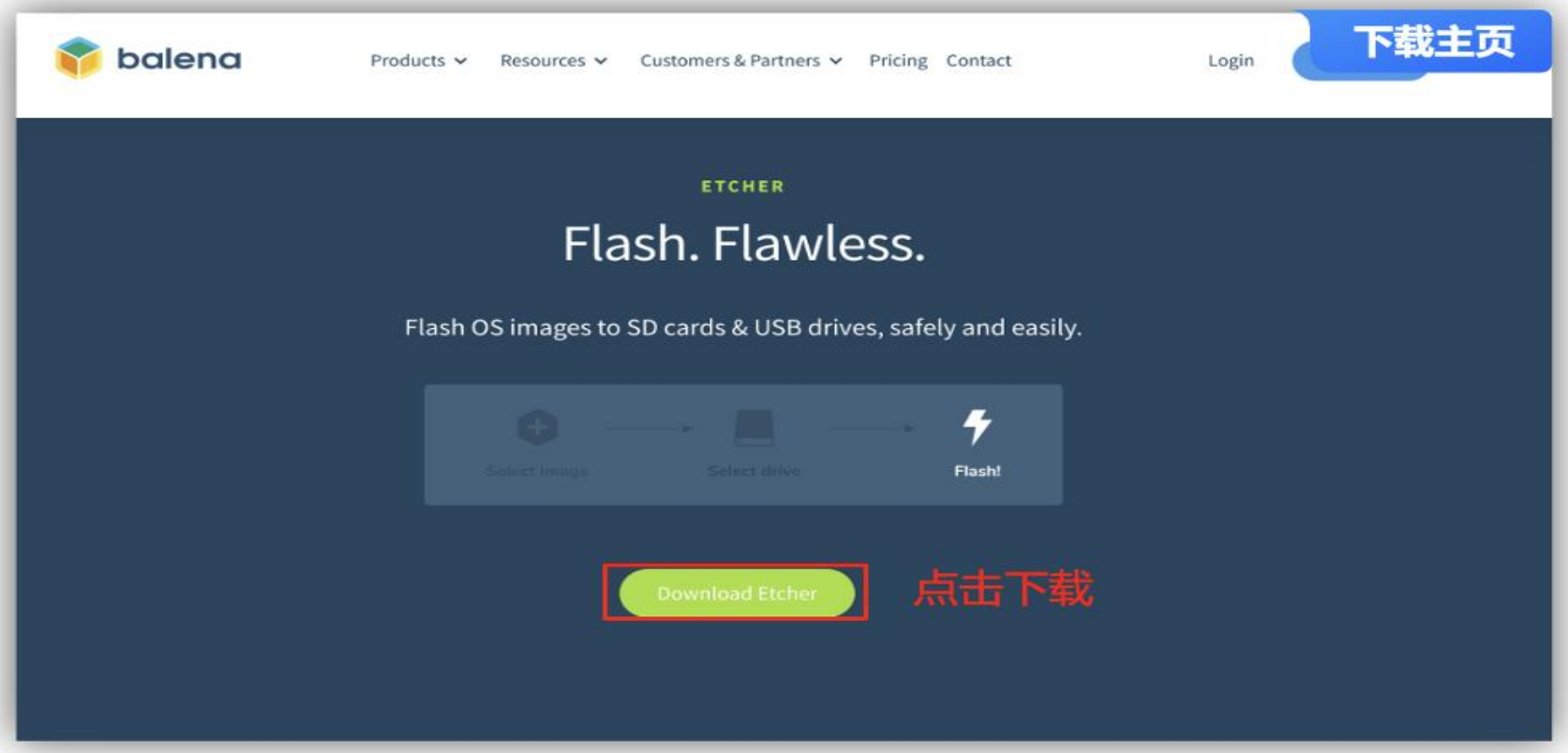
环境搭建

开发板资料下载页面的链接如下所示：

http://www.orangepi.cn/html/hardWare/computerAndMicrocontrollers/service-and- support/Orange-Pi-AIpro.html

环境搭建

首先打开balena烧录工具地址，我们选择OS为“WINDOWS”的X86|X64版本下载。



环境搭建

打开下载的balena烧录工具，不用安装，直接就打开了，非常的方便，不用在电脑上安装软件：

点击“从文件烧录”选择好我们提前下载的镜像

点击“选择目标磁盘”选中我们插入的TF卡

点击“现在烧录”就可以开始进行烧录了

机器组装

插入USB的鼠标和键盘、显示器HDMI线、自带的type-c电源线后，即可插上电源了，插一通电就自动启动。

启动时，需要输入一个用户名密码。

用户名：HwHiAiUser

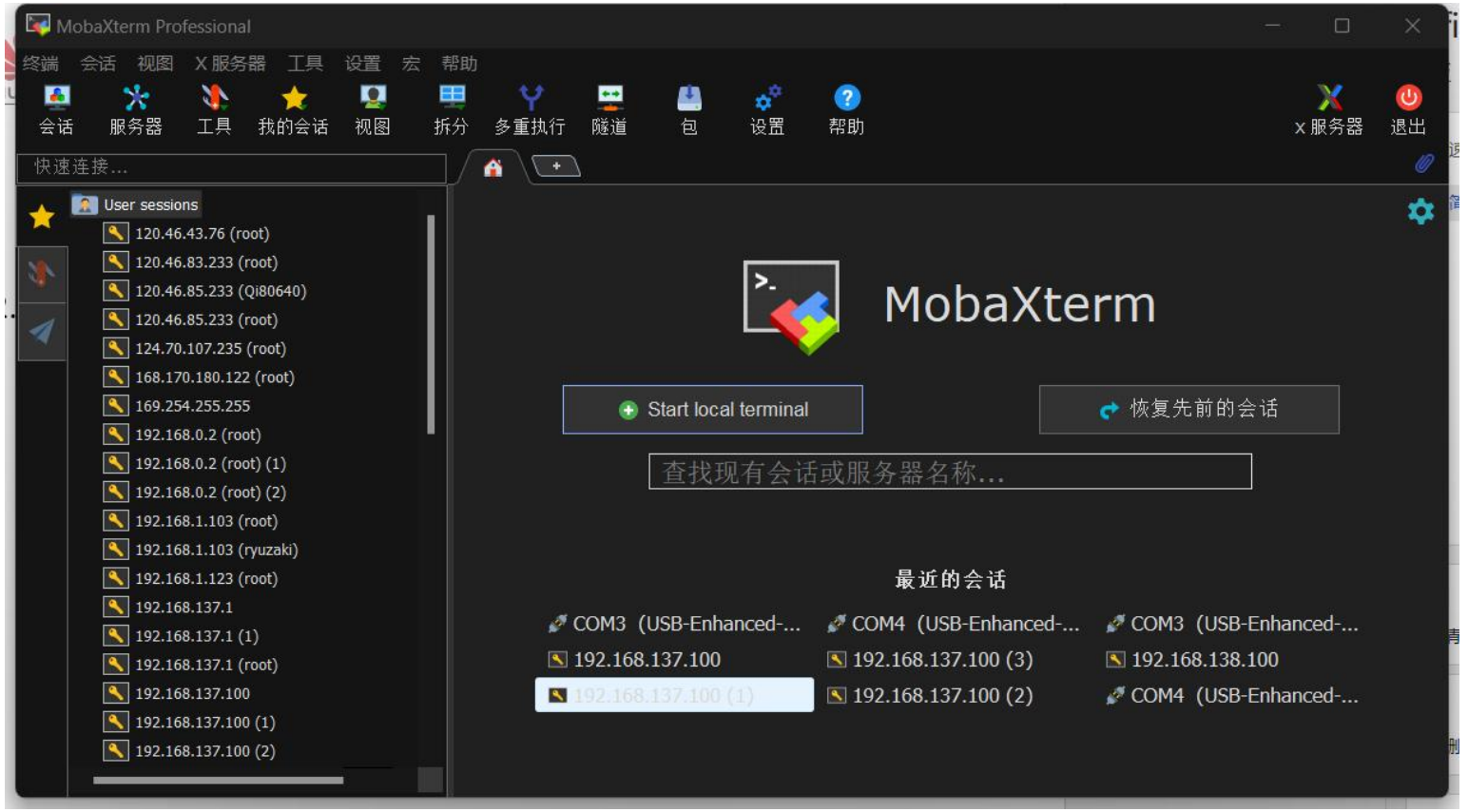
密码：Mind@123

序号	可能存在问题点	解决方案	难度
1	检查一下控制启动设备的两个拨码开关，要想从TF卡启动，需要都是右	检查无问题	1星
2	可能烧录的时候出现上面异常，导致无法亮灯	可以尝试使用网友提供的ascend重新烧录一下	2星
3	手里是否有备用TF卡，不过最好使用容量大点的，32G以上，推荐64G	需要买一张备用卡	3星
4	可能开发板子存在问题	这个需要专业人员来解决	4星

串口登录

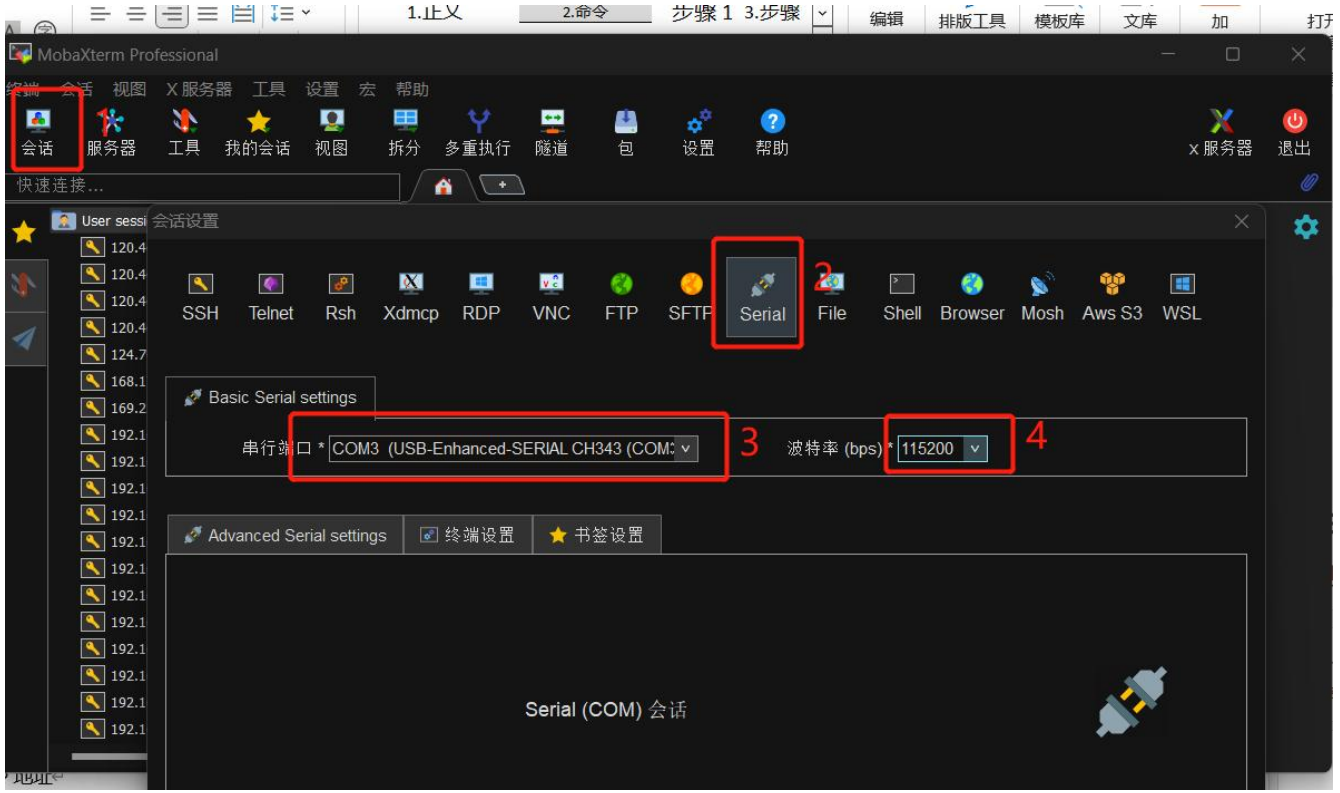
下载mobaxtem

安装并启动



串口登录

点击1.会话->2.serial->3.选择对应的com口->4.选择波特率为115200



串口登录

1.按回车按钮, 2.HwHiAiUser,3.密码Mind@123

链接Wifi，使用SSH登录

在同一个局域网，使用ping ip地址保证网是能连通的情况下，就可以使用SSH登录到香橙派上

```
# 查看主机IP地址
ifconfig

# 检查网络是否连通
ping 192.xx.xx.xx
```

需要升级的内容

CANN升级

Toolkit升级

Kernels升级

MindSpore升级

环境搭建指南：https://www.mindspore.cn/docs/zh-CN/r2.4.10/orange_pi/environment_setup.html

测试 MindSpore 的方法

开发板的 Ubuntu 系统中已经预装了 MindSpore，使用下面的方法可以测试下 MindSpore 是否能正常使用。

在 HwHiAiUser 用户中执行下面的命令：

```
(base) HwHiAiUser@orangepiaipro:~$ python -c \ "import
mindspore;mindspore.set_context(device_target='Ascend');mindspore.run_check()"
```

等待一段时间后，如果能看到下面的输出就说明 MindSpore 能正常使用。

```
MindSpore version: 2.x.xx.2024xxxx
The result of multiplication calculation is correct, MindSpore has been installed on platform
[Ascend] successfully!
```

基于MindSpore的手写数字识别

通过基于MindSpore的手写数字识别案例，说明香橙派开发板中的开发注意事项。

设置运行环境

```
import mindspore
mindspore.set_context(max_device_memory="2GB",
                      pynative_synchronize=True,
                      mode=mindspore.GRAPH_MODE,
                      device_target="Ascend",
                      jit_config={"jit_level":"O2"},
                      ascend_config={"precision_mode":"allow_mix_precision"})
```

max_device_memory=“2GB”：设置设备可用的最大内存为2GB。

mode=mindspore.GRAPH_MODE：表示在GRAPH_MODE模式中运行。

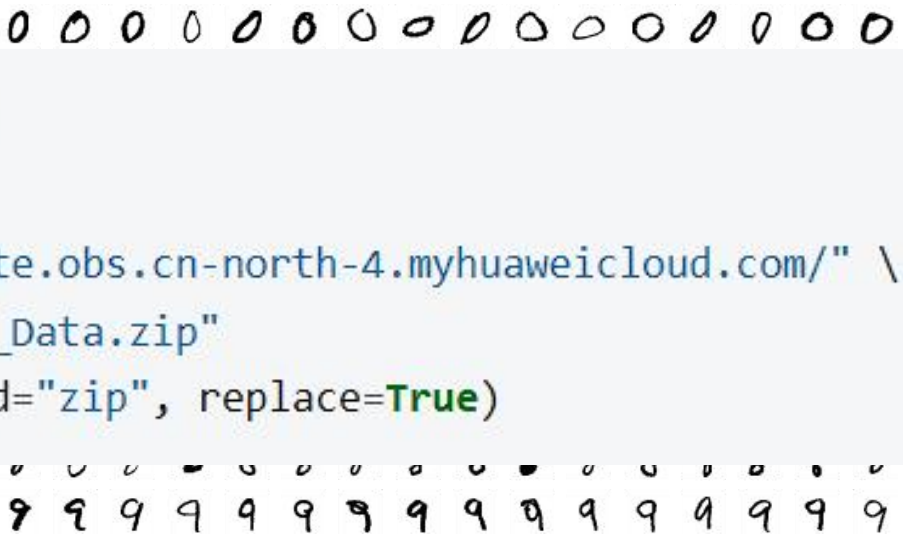
device_target=“Ascend”：表示待运行的目标设备为Ascend。

jit_config={“jit_level”：“O2”}：编译优化级别开启极致性能优化，使用下沉的执行方式。

scend_config={“precision_mode”：“allow_mix_precision”}：自动混合精度，自动将部分算子的精度降低到float16或bfloat16。

处理数据集

MindSpore提供基于Pipeline的数据引擎，通过数据加载与处理实现高效的数据预处理。在本案例中，我们使用Mnist数据集，自动下载完成后，使用mindspore.dataset提供的数据变换进行预处理。



```
# Download data from open datasets
from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/" \
      "notebook/datasets/MNIST_Data.zip"
path = download(url, "./", kind="zip", replace=True)
```

处理数据集

MNIST数据集目录结构如下：

```
MNIST_Data
└── train
    ├── train-images-idx3-ubyte (60000个训练图片)
    ├── train-labels-idx1-ubyte (60000个训练标签)
└── test
    ├── t10k-images-idx3-ubyte (10000个测试图片)
    ├── t10k-labels-idx1-ubyte (10000个测试标签)
```

处理数据集

数据下载完成后，获得数据集对象。

```
train_dataset = MnistDataset('MNIST_Data/train')
test_dataset = MnistDataset('MNIST_Data/test')
```

打印数据集中包含的数据列名，用于dataset的预处理。

```
print(train_dataset.get_col_names())
```

```
['image', 'label']
```

处理数据集

MindSpore的dataset使用数据处理流水线（Data Processing Pipeline），需指定map、batch、shuffle等操作。这里我们使用map对图像数据及标签进行变换处理，将输入的图像缩放为1/255，根据均值0.1307和标准差值0.3081进行归一化处理，然后将处理好的数据集打包为大小为64的batch。

```python
def datapipe(dataset, batch_size):
    image_transforms = [
        vision.Rescale(1.0 / 255.0, 0),
        vision.Normalize(mean=(0.1307,), std=(0.3081,)),
        vision.HWC2CHW()
    ]
    label_transform = transforms.TypeCast(mindspore.int32)

    dataset = dataset.map(image_transforms, 'image')
    dataset = dataset.map(label_transform, 'label')
    dataset = dataset.batch(batch_size)
    return dataset
```

```python
# Map vision transforms and batch dataset
train_dataset = datapipe(train_dataset, 64)
test_dataset = datapipe(test_dataset, 64)
```

处理数据集

可使用create_tuple_iterator 或create_dict_iterator对数据集进行迭代访问，查看数据和标签的shape和datatype。

以create_tuple_iterator为例：

```
for image, label in test_dataset.create_tuple_iterator():
    print(f"Shape of image [N, C, H, W]: {image.shape} {image.dtype}")
    print(f"Shape of label: {label.shape} {label.dtype}")
    break
```

```
Shape of image [N, C, H, W]: (64, 1, 28, 28) Float32
Shape of label: (64,) Int32
```

构建模型

- mindspore.nn类是构建所有网络的基类，也是网络的基本单元。当用户需要自定义网络时，可以继承nn.Cell类，并重写__init__方法和construct方法。__init__包含所有网络层的定义，construct中包含数据（Tensor）的变换过程。

```
# Define model
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.dense_relu_sequential = nn.SequentialCell(
            nn.Dense(28*28, 512),
            nn.ReLU(),
            nn.Dense(512, 512),
            nn.ReLU(),
            nn.Dense(512, 10)
        )

    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits

model = Network()
print(model)
```

```
Network<
  (flatten): Flatten<>
  (dense_relu_sequential): SequentialCell<
    (0): Dense<input_channels=784, output_channels=512, has_bias=True>
    (1): ReLU<>
    (2): Dense<input_channels=512, output_channels=512, has_bias=True>
    (3): ReLU<>
    (4): Dense<input_channels=512, output_channels=10, has_bias=True>
    >
  >
```

模型训练

推荐博文：PyTorch基础完整模型训练套路（土堆老师版）详细注释及讲解！小白学习必看！

在模型训练中，一个完整的训练过程（step）需要实现以下三步：

1. **正向计算**：模型预测结果（logits），并与正确标签（label）求预测损失（loss）。
2. **反向传播**：利用自动微分机制，自动求模型参数（parameters）对于loss的梯度（gradients）。
3. **参数更新**：将梯度更新到参数上。

MindSpore使用函数式自动微分机制，因此针对上述步骤需要实现：

① 定义正向计算函数。

② 使用value_and_grad通过函数变换获得梯度计算函数。

定义训练函数，使用set_train设置为训练模式，执行正向计算、反向传播和参数优化。

模型训练

除训练外，我们定义测试函数，用来评估模型的性能。

```
def test(model, dataset, loss_fn):
    num_batches = dataset.get_dataset_size()
    model.set_train(False)
    total, test_loss, correct = 0, 0, 0
    for data, label in dataset.create_tuple_iterator():
        pred = model(data)
        total += len(data)
        test_loss += loss_fn(pred, label).asnumpy()
        correct += (pred.argmax(1) == label).asnumpy().sum()
    test_loss /= num_batches
    correct /= total
    print(f"Test: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

模型训练

训练过程需多次迭代数据集，一次完整的迭代称为一轮（epoch）。在每一轮，遍历训练集进行训练，结束后使用测试集进行预测。打印每一轮的loss值和预测准确率（Accuracy），可以看到loss在不断下降，Accuracy在不断提高。

```
Epoch 1
-------------------------------
loss: 2.302898  [  0/938]
loss: 1.729961  [100/938]
loss: 0.865714  [200/938]
loss: 0.782822  [300/938]
loss: 0.389282  [400/938]
loss: 0.293149  [500/938]
loss: 0.474819  [600/938]
loss: 0.242542  [700/938]
loss: 0.542277  [800/938]
loss: 0.342929  [900/938]
Test:
 Accuracy: 90.7%, Avg loss: 0.321954
```

```
Epoch 2
-------------------------------
loss: 0.249492  [  0/938]
loss: 0.347967  [100/938]
loss: 0.220382  [200/938]
loss: 0.308149  [300/938]
loss: 0.353044  [400/938]
loss: 0.392116  [500/938]
loss: 0.396438  [600/938]
loss: 0.231412  [700/938]
loss: 0.194819  [800/938]
loss: 0.228290  [900/938]
Test:
 Accuracy: 93.0%, Avg loss: 0.249993
```

```
Epoch 3
-------------------------------
loss: 0.343888  [  0/938]
loss: 0.307786  [100/938]
loss: 0.153425  [200/938]
loss: 0.254917  [300/938]
loss: 0.198072  [400/938]
loss: 0.108963  [500/938]
loss: 0.202033  [600/938]
loss: 0.340418  [700/938]
loss: 0.144911  [800/938]
loss: 0.175447  [900/938]
Test:
 Accuracy: 93.7%, Avg loss: 0.212180
```

保存模型

模型训练完成后，需要将其参数进行保存。

```
# Save checkpoint
mindspore.save_checkpoint(model, "model.ckpt")
print("Saved Model to model.ckpt")
```

加载模型

加载保存的权重分为两步：

① 重新实例化模型对象，构造模型。

② 加载模型参数，并将其加载至模型上。

```
# Instantiate a random initialized model
model = Network()
# Load checkpoint and load parameter to model
param_dict = mindspore.load_checkpoint("model.ckpt")
param_not_load, _ = mindspore.load_param_into_net(model, param_dict)
print(param_not_load)
```

param_not_load是未被加载的参数列表，为空时代表所有参数均加载成功。 `[]`

加载模型

加载后的模型可以直接用于预测推理。

```
model.set_train(False)
for data, label in test_dataset:
    pred = model(data)
    predicted = pred.argmax(1)
    print(f'Predicted: "{predicted[:10]}", Actual: "{label[:10]}"')
    break
```

```
Predicted: "[1 2 0 4 6 4 9 0 2 2]", Actual: "[1 2 0 4 6 9 9 0 2 2]"
```

总结

上面我们通过MindSpore的API来快速实现一个简单的深度学习模型，从加载数据集、数据变换 Transforms、网络构建、模型训练、模型保存、加载模型，完成了一个简单的深度学习模型的闭环，可以体验到：

①能过jupyter云上开发环境，我们可以不用搭建自己的AI应用。

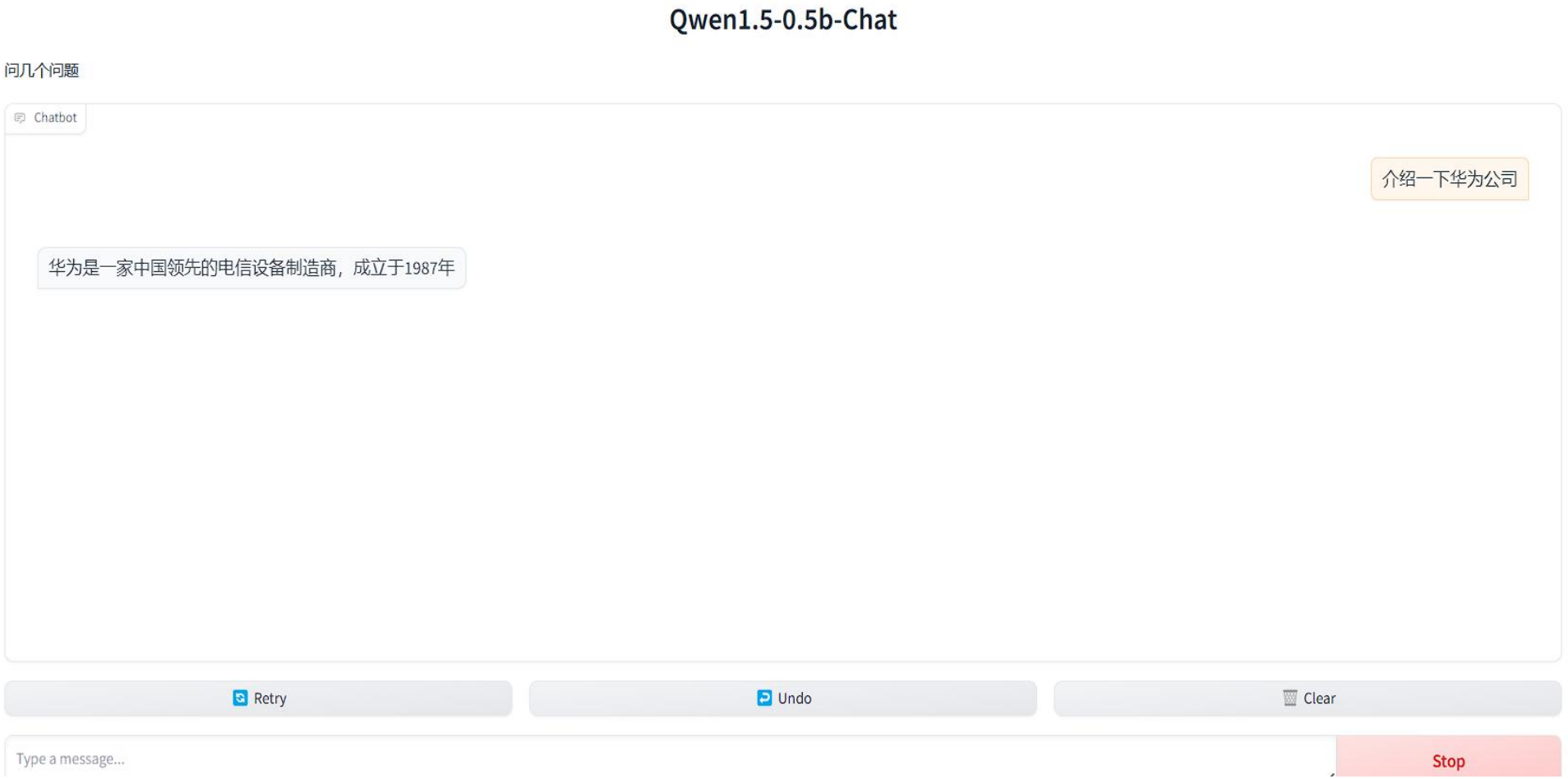
②MindSpore的API简化了大量深度学习的复杂度。

③通过使用华为云可以达到百M速度的下载，十分的高效。

④在模型训练中，可以在几分钟内就完成模型的训练，大大的简化了实验中等待的过程。

代码：https://www.mindspore.cn/docs/zh-CN/r2.4.10/orange_pi/dev_start.html

基于mindnlp的qwen1.5-0.5b聊天机器人部署



导入相关库

```
import gradio as gr
import mindspore
from mindnlp.transformers import AutoModelForCausalLM, AutoTokenizer
from mindnlp.transformers import TextIteratorStreamer
from threading import Thread
```

加载tokenizer和model

```
# Loading the tokenizer and model from Hugging Face's model hub.
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen1.5-0.5B-Chat", ms_dtype=mindspore.float16)
model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen1.5-0.5B-Chat", ms_dtype=mindspore.float16)
```

如遇网络问题无法加载参数可先将模型权重下载至本地

从历史聊天记录构建输入

```
def build_input_from_chat_history(chat_history, msg: str):
    messages = [{'role': 'system', 'content': system_prompt}]
    for user_msg, ai_msg in chat_history:
        messages.append({'role': 'user', 'content': user_msg})
        messages.append({'role': 'assistant', 'content': ai_msg})
    messages.append({'role': 'user', 'content': msg})
    return messages
```

推理函数

```
# Function to generate model predictions.
def predict(message, history):
    history_transformer_format = history + [[message, ""]]

    # Formatting the input for the model.
    messages = build_input_from_chat_history(history, message)
    input_ids = tokenizer.apply_chat_template(
            messages,
            add_generation_prompt=True,
            return_tensors="ms",
            tokenize=True
        )
    eamer = TextIteratorStreamer(tokenizer, timeout=120, skip_prompt=True, skip_special_tokens=True)
    erate_kwargs = dict(
        input_ids=input_ids,
        streamer=streamer,
        ax_new_tokens=1024,
        o_sample=True,
        op_p=0.9,
        temperature=0.1,
        num_beams=1,
    )
     Thread(target=model.generate, kwargs=generate_kwargs)
    tart()  # Starting the generation in a separate thread.
    tial_message = ""
    for new_token in streamer:
        partial_message += new_token
        if '</s>' in partial_message:  # Breaking the loop if the stop token is generated.
            break
        yield partial_message
```

构建gradio交互界面

```
# Setting up the Gradio chat interface.
gr.ChatInterface(predict,
                 title="Qwen1.5-0.5b-Chat",
                 description="问几个问题",
                 examples=['你是谁？', '介绍一下华为公司']
                 ).launch()  # Launching the web interface.
```

构建了上面P30的对话界面

作业

作业内容：基于MindNLP套件迁移mimi模型。

参考资料

- 参考代码仓：https://github.com/huggingface/transformers/tree/main/src/transformers/models/mimi
- MindNLP仓库：https://github.com/mindspore-lab/mindnlp

验收标准

提交内容：Configuration，Model，Unit tests

提交PR至MindNLP代码仓 https://github.com/mindspore-lab/mindnlp

要求通过门禁测试

huggingface全量u 通过