南京大学

智 能 计 算 系 统
实 验 操 作 手 册

实 验 序 号	三
实 验 名 称	基于昇腾硬件端侧部署大语言模型
院　　系	计算机学院

2025 年 11 月 6 日

目　录

实验三　基于昇腾硬件端侧部署大语言模型

3.1　昇腾硬件介绍

昇腾（Ascend）系列 NPU 是华为自研的 AI 加速器产品线，基于达芬奇（Da Vinci）架构，支持 FP16、INT8 等混合精度与算子加速，面向从边缘到数据中心的推理与训练。其中 Ascend 310/310B 系列芯片偏向端侧与边缘推理，强调低功耗与能效比，常用于摄像头、网关、机器人等；Ascend 910/910B 系类芯片面向数据中心训练与高性能推理，可支撑更大规模模型与更高吞吐。

除了硬件和芯片之外，昇腾还提供了完善的软件生态，包括 CANN（算子与编译器栈）、AscendCL/ACL API、Ascend Toolchain（ATC/AOE 模型转换与优化）、以及与 MindSpore 框架的深度适配。

本次实验所使用的香橙派（Orange Pi）AI Pro 开发板集成了华为昇腾 310B 系列 NPU。并且提供了丰富的接口，包括摄像头、显示（如 HDMI）、千兆以太网、USB、GPIO，以及 M.2/PCIe 扩展等。

3.2　开发板使用指南

下面介绍几种连接和使用开发版的常用方式。

3.2.1　显示器连接（推荐）

1、除了香橙派开发板之外需要自行准备的设备如下：

- 一个显示器。
- 一条双向 HDMI 线，用于连接开发板与显示器。
- 一幅 USB 键盘和鼠标，用于操作开发板。

2、使用步骤：

1. 连接显示器、键盘和鼠标之后，连接电源，等待开机。
2. 连接电源后风扇会高速转动，在风扇停止转动之后不久，开机过程完成，屏幕上应该会显示登录界面。这一步的时间可能会有点长。
3. 输入账号：HwHiAiUser，密码：Mind@123 后进入系统。(可能你的开发板启动之后没有要求输入密码，这是正常的情况)

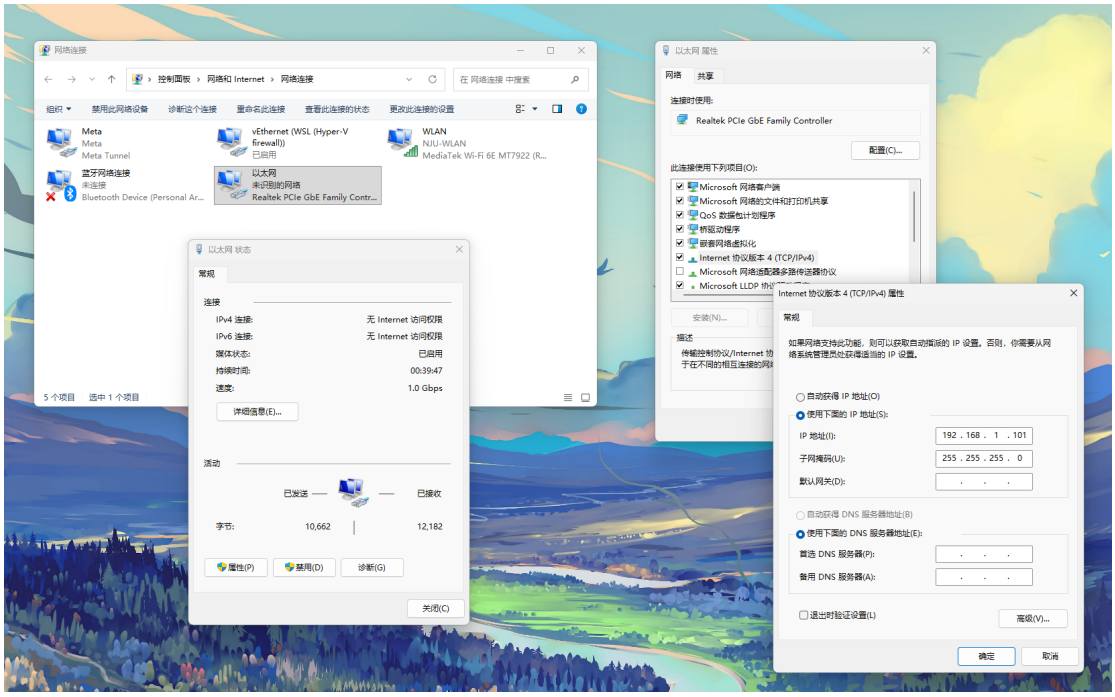
3.2.2 网线直连

1、除了香橙派开发板之外需要自行准备的设备如下：

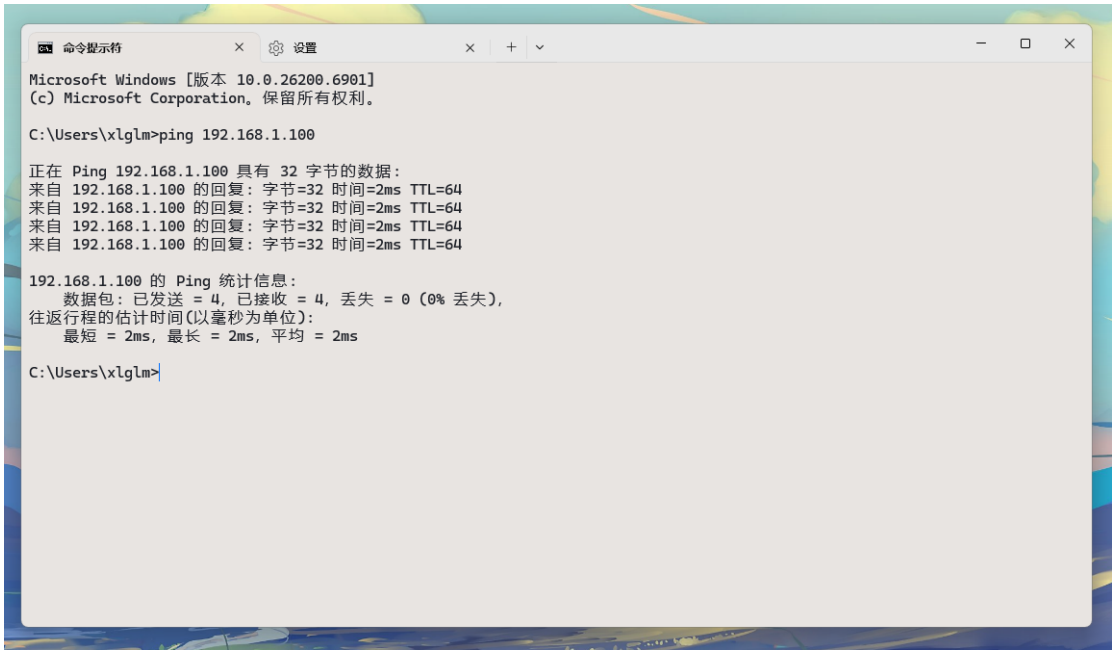
- 一台能连接网线的电脑。
- 一条双头网线。

2、使用步骤（以 Windows 系统为例）：

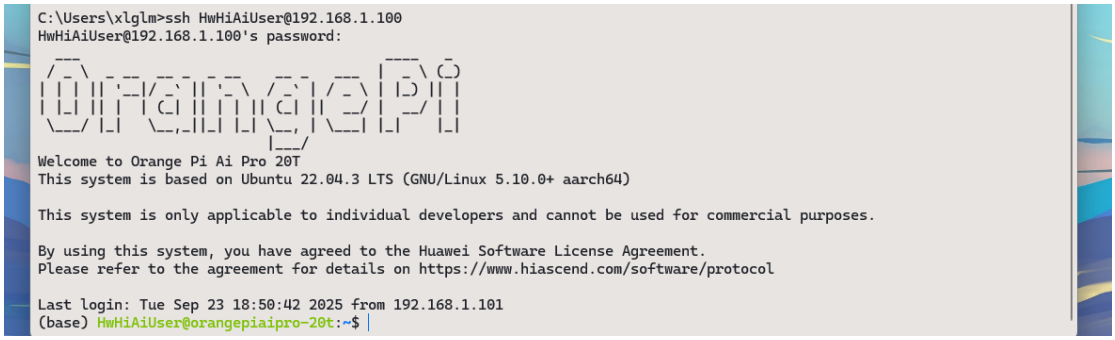
1. 用网线连接笔记本和开发板，（请使用开发板上靠近电源的那一个网线插口，两个网线插口的作用不同）。
2. 配置本地以太网地址，将本地以太网接口的 ip 地址改为 192.168.1.101（如图所示）。



3. 开发板的 IP 地址为 192.168.1.100，打开终端，如 powershell/cmd，使用 ping 命令确认连接成功。



4. 在终端中，使用 ssh 连接到开发板，用户名为 HwHiAiUser，密码为 Mind@123，（命令 ssh HwHiAiUser@orange-ip）。



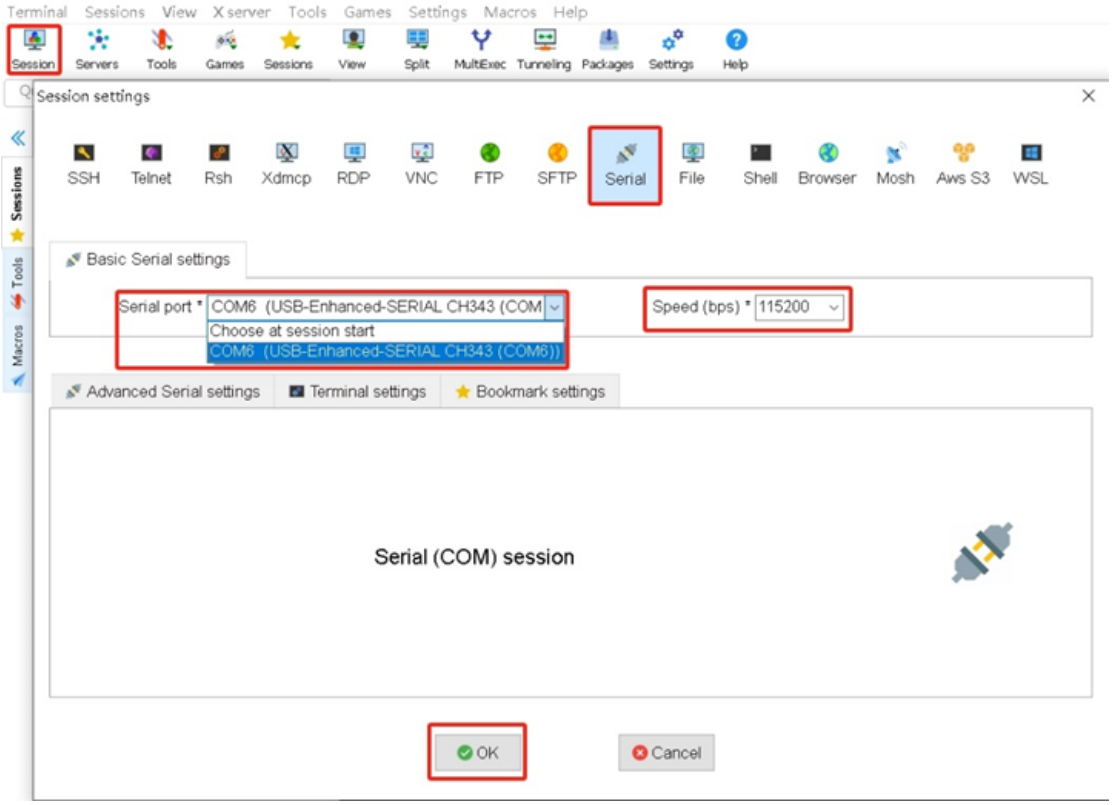
3.2.3 USB 串口连接（推荐）

1、除了香橙派开发板之外需要自行准备的设备如下：

- 一台有 USB/Type-C 接口的电脑（Windows/MacOS/Linux 系统不限，以下以 Windows 系统为例）。
- 笔记本上下载 mobaxterm/Tabby 等可以连接串口的终端软件（以下以 mobaxterm 为例）。
- 一条 USB-typeC 的数据线。

2、使用步骤

1. 用数据线连接笔记本的 USB 插口和开发板的 UART 插口。
2. 打开 mobaxterm，开启一个新的 session，选择 serial，Serial Port 栏下拉选择 CH343，波特率选择 115200。



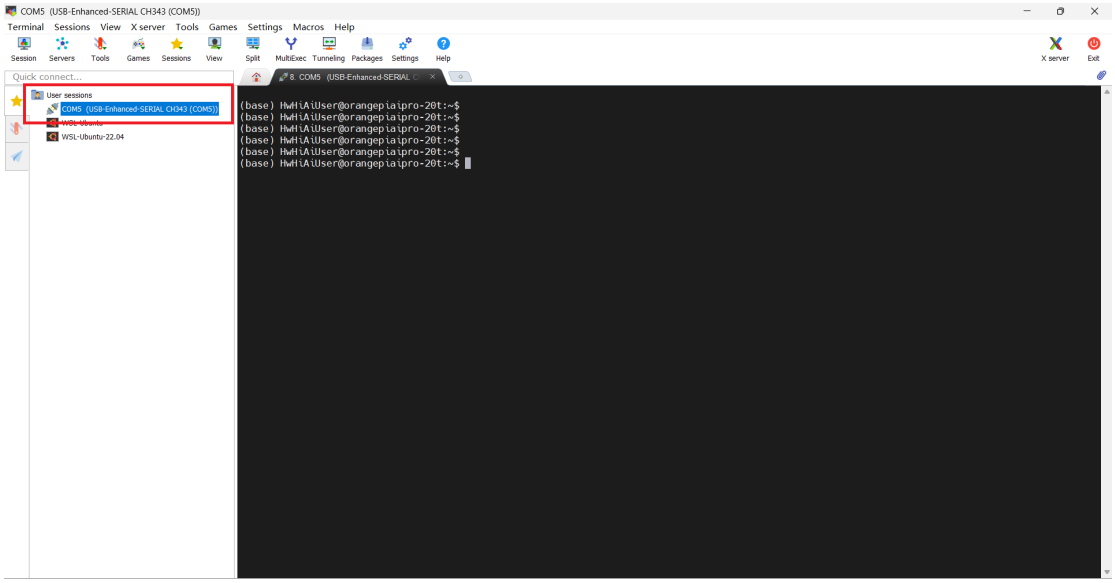
3. 如果你没有找到 CH343，请尝试安装我们提供的 USB 转 TTL 串口芯片的驱动。
4. 点击 OK 后，开发板上电，就会进入登录界面，输入账号 HwHiAiUser，密码 Mind@123，即可进入实验环境。
5. 下次登陆时，在连接开发板的情况下，可以直接点击左侧的记录，快速登录。

3.2.4 以太网连接

此方法需要基于前三种方法中的一种进行操作，否则无法获知开发板的以太网（校园网）IP 地址。此方法的优点在于，在香橙派连接电源和网线后，可以将其当作服务器使用，通过校园网即可远程登录和操作。

1、在使用上述三种方式中的一种登录香橙派之后，需要额外准备的设备有：

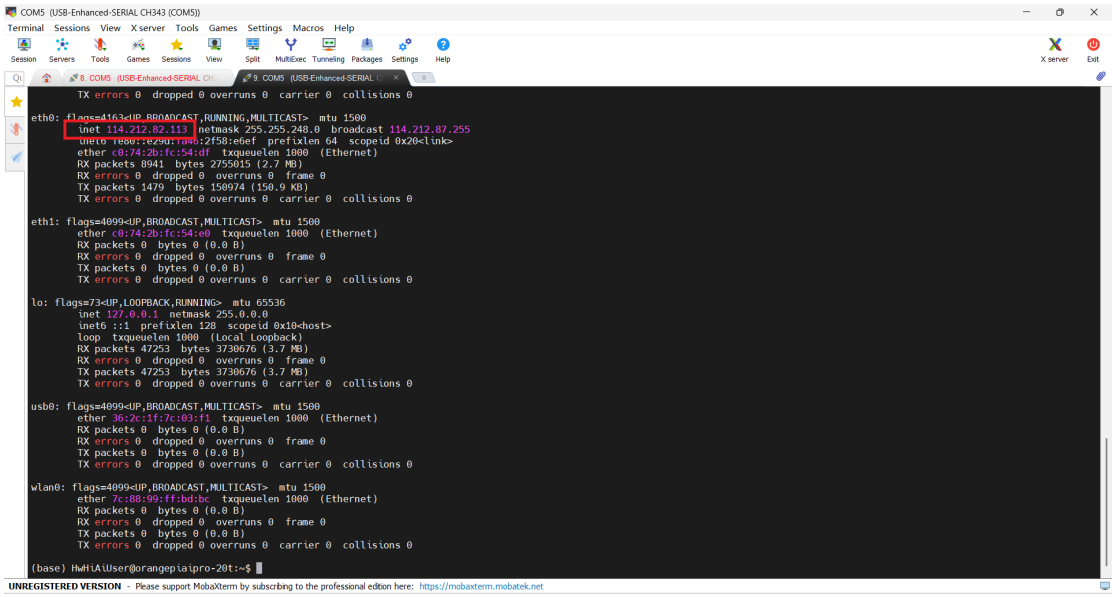
- 一条网线。



- 校园网网口（通常，宿舍的每个人位置的墙上插座附近都有一个）。

2、使用步骤：

1. 香橙派上电开机后，将网线插入香橙派上的网口 2（远离电源的那个）和另一端的校园网网口。
2. 待完成开机后，使用前述三种方式之一登录香橙派。
3. 进入到香橙派的终端，使用 ifconfig 命令，查看分配的校园网 IP。



4. 保持香橙派的电源和网线连接，后续便可以用 ssh 远程登陆到香橙派上，账户密码与前述相同。ssh HwHiAiUser@xxx.xxx.xxx

5. 正常情况下，香橙派的 IP 是不会发生改变的，即便重新开机或者换个校园网网口（DHCP 会让它继续使用），如果发现远程无法登录的情况，请查看香橙派是否正常运行，以及 IP 地址是否发生变化。

3.2.5 注意事项

1、香橙派（及其 tf 卡）、快充头、数据线等设备教学使用后还需要回收，请尽量保护好设备，但是如果出现什么问题还是可以和助教或老师沟通，我们会积极协助解决，无须有过多的心理负担；

2、如果上电后风扇不转灯不亮，请注意供电口是最靠近边上的 type-C 口（标记了 DC Direct Current），插到中间的口的话是供不上电的；上电后，系统开始自检时，风扇会以最大速度转动，进入系统后，风扇转速会降低，此时可以建立 ssh 连接了；

3、如无特殊需求，请勿清除 tf 卡（香橙派背面插着的小闪存卡）内的系统，如果有特殊需求，请联系助教或老师；

4、tf 卡烧录的文件系统在 Windows 环境下是无法直接读取的（可以使用 Linux 环境，或使用 DiskGenius 等软件），并且会弹出格式化提示，注意不要格式化。

3.3 实验目的

本实验旨在掌握大语言模型（Large Language Model，LLM）在昇腾（Ascend）硬件平台上的端侧部署流程与优化方法，理解从预训练模型到端侧推理的关键环节，包括模型压缩、算子优化、ONNX 转换，以及生成与执行昇腾离线模型（OM）的全过程。

通过本实验，学生将深入理解在算力受限的端侧设备及昇腾系列 NPU 上运行大语言模型的优化策略与机制，掌握从框架级导出到硬件执行的完整部署流程，并能根据端侧硬件的特性（如算力、内存约束等）设计针对性的优化方案。

本实验主要目的如下：

1. 理解大语言模型的基本组成与推理流程。掌握 Transformer 架构在 LLM 中的应用，了解推理阶段的关键计算模块（如注意力机制、前馈网络、位置

编码等）。

2. 掌握在昇腾端侧的模型部署流程。学习如何将开源预训练大模型（如 LLaMA）导出为 ONNX 格式，并使用昇腾工具链（如 ATC）将其转换为可在端侧运行的离线模型（OM 模型）。
3. 掌握端侧推理与性能评估方法。学会在昇腾 NPU 设备上执行推理任务，监控关键性能指标（如延迟、吞吐量、内存占用），并验证模型在端侧的正确性和效率。
4. 理解模型压缩与量化在端侧部署中的作用。掌握以量化为代表的模型优化方法，理解这些方法在降低计算复杂度和存储需求方面的意义。

本实验将在搭载昇腾 310B NPU 的香橙派 AI Pro 上开展。实验将依次完成模型量化、导出为 ONNX 格式、使用 ATC（Ascend Tensor Compiler）编译，并部署大模型以进行推理。

3.4 背景介绍

随着大语言模型（Large Language Model，LLM）的广泛应用，其参数规模和计算复杂度呈指数级增长。由此，模型在推理阶段对算力、存储与带宽的需求也显著上升。传统上，LLM 的推理主要依赖云端 GPU 集群；要在资源受限的端侧设备上部署大型模型，则需要结合模型压缩、算子优化和硬件加速等技术。

3.4.1 昇腾端侧部署与香橙派 AI Pro 20T 平台

昇腾（Ascend）是华为推出的一系列人工智能计算平台，覆盖云端到端侧的多种算力场景。昇腾 20T 是面向端侧推理的高能效 NPU 芯片，具有强大的矩阵计算能力和片上算子优化机制。香橙派 AI Pro 20T 是基于昇腾 20T 的开发平台，内置 CANN（Compute Architecture for Neural Networks）软件栈，支持从模型编译到推理执行的完整流程。

该平台支持将主流深度学习框架（如 PyTorch、TensorFlow、HuggingFace Transformers）导出的 **ONNX 模型**转换为可在 NPU 上运行的 **OM 模型**。平台提供 ATC（Ascend Tensor Compiler）用于图优化和算子融合，从而提升模型在端侧的执行效率。

3.4.2 TinyLlama 模型简介

TinyLlama 是基于 Llama 架构的轻量化大语言模型，旨在在有限硬件资源下实现较好的语言理解与生成能力。该模型通过减少层数与参数量、调整隐藏维度、共享权重等方式，降低了计算与存储开销。TinyLlama 保留了 Transformer 解码器结构的核心部分，包括多头自注意力与前馈网络模块，同时适配低精度计算与量化推理，为在端侧设备上的高效部署提供了良好基础。

在本实验中，我们选用 TinyLlama 作为基础模型。首先通过 **HuggingFace Transformers** 框架加载预训练权重，并将模型导出为 ONNX 格式；随后使用昇腾工具链将 ONNX 转换为 OM 模型，最优将 OM 模型部署到香橙派 AI Pro 20T 上进行端侧推理。

3.4.3 模型量化与端侧优化技术

为了在端侧设备上高效运行大语言模型，必须降低模型在推理时的计算与存储需求。模型量化（Quantization）是一种常用的优化方法，其核心思想是将原本使用 32 位浮点数（FP32）表示的权重和激活值压缩为较低位宽（如 INT8 或 FP16），以显著减少模型的存储空间与运算量。

根据是否依赖训练过程，模型量化可分为两类：

- **训练后量化（Post-Training Quantization, PTQ）**：在模型训练完成后直接进行权重量化，通常通过校准数据集估计量化尺度（scale）和零点（zero point），无需重新训练模型，部署效率高，适用于端侧快速验证场景。
- **量化感知训练（Quantization-Aware Training, QAT）**：在训练阶段显式模拟量化过程，使模型在量化误差下仍能保持较高精度。该方法适合对精度要求较高的场景。

本次实验中（包括实际环境中的大部分情况），将使用简单高效的 PTQ 作为模型量化方案。

量化在端侧部署中的优势主要体现在：

1. **减少存储需求**：通过降低数值精度显著压缩模型参数体积；
2. **提升推理速度**：NPU 可利用低精度计算单元并行加速；

3. **降低能耗**：减少内存带宽与算力消耗，有助于端侧持续运行；
4. **提高硬件适配性**：支持昇腾 NPU 对 INT8、FP16 精度的高效执行。

综上，量化技术在大语言模型端侧部署中具有重要意义。通过结合昇腾硬件的异构加速与 TinyLlama 的轻量化结构，可以在资源受限的香橙派 AI Pro 20T 平台上实现高效的大语言模型推理。

3.5 实验环境

设备：搭载昇腾 310B NPU 的香橙派 AI pro 20T

操作系统：Ubuntu 22.04

环境依赖：

表 3-1 环境依赖

名称	版本
Python	== 3.9
transformers	== 4.35
torch	== 2.1.0
CANN	>= 8.0

3.6 实验内容

首先对 `TinyLlama-chat-v1.0` 模型进行量化，并导出为 ONNX 格式。该模型包含完整的推理图结构，能够在低精度（如 INT8 或 FP16）下运行。利用昇腾工具链中的 ATC（Ascend Tensor Compiler）工具，将 ONNX 模型编译为可在昇腾 NPU 上执行的 `OM` 离线模型文件。编译完成后将生成 `tinyllama-chat-v1.0.om` 文件，用于端侧加载与推理。在香橙派 AI Pro 20T 上自行编写 Python 推理脚本，加载编译生成的 OM 模型，完成从输入文本到生成回复的完整流程。

本实验包括，模型量化，ONNX 模型导出，OM 模型编译，模型部署和推理。

3.7 实验步骤

本实验旨在完成基于昇腾硬件的端侧大语言模型部署与推理,使用 `Tinyllama-chat-v1.0` 模型为基础，量化后，通过昇腾工具链进行模型编译与部署，最终在香橙派 AI Pro 20T 平台上实现对话式推理功能。实验的主要流程如下：

3.7.1 步骤一：准备环境

protoc 安装

```
wget -O protobuf-all-3.13.0.tar.gz https :// box.nju.edu.cn/f/7
    ↪ c928229250d48dfafea/?dl=1
tar -zxvf protobuf-all-3.13.0.tar.gz
cd protobuf-3.13.0
apt-get update
apt-get install autoconf automake libtool
./autogen.sh
./configure
make -j4
make install
sudo ldconfig
protoc --version
```

算子编译部署

```
cd tiny_llama
export ASCEND_PATH=/usr/local/Ascend/ascend-toolkit/latest
cp custom_op/matmul_integer_plugin.cc $ASCEND_PATH/tools/msopgen/
    ↪ template/custom_operator_sample/DSL/Onnx/framework/onnx_plugin/
cd $ASCEND_PATH/tools/msopgen/template/custom_operator_sample/DSL/
    ↪ Onnx
```

打开 build.sh，找到下面四个环境变量，解开注释并修改如下：

```
export ASCEND_TENSOR_COMPILER_INCLUDE=/usr/local/Ascend/ascend
    ↪ -toolkit/latest/include
export TOOLCHAIN_DIR=/usr
export AICPU_KERNEL_TARGET=cust_aicpu_kernels
export AICPU_SOC_VERSION=Ascend310B4
```

```
./build.sh
cd build_out/
./custom_opp_ubuntu_aarch64.run
cd \$ASCEND_PATH/opp/vendors/customize
rm -rf op_impl/ op_proto/
```

3.7.2 步骤二：量化模型

实验中采用 **W8A8 量化方案**（即权重和激活值均采用 8 位整数表示），以降低模型存储与计算开销。

设原始张量为 x，量化后的整数表示为 $\hat{x}$，其量化与反量化过程可表示为：

$$\hat{x} = \text{round}\left(\frac{x}{s}\right) + z, \quad x \approx s \times (\hat{x} - z)$$

其中 s 为缩放因子（scale），z 为零点（zero point）。通过对权重与激活值分别统计范围，可获得量化所需的尺度参数。

权重量化可直接依据参数分布计算，而激活值量化需通过模型的前向推理过程动态收集。

使用少量数据获取激活值

为了计算各层输入激活值的最大幅度（用于确定 scale），使用少量样本（如 wikitext 数据集）进行推理统计。具体实现如下代码所示，通过为线性层注册前向钩子（forward hook）来收集激活值分布：

```
def get_act_scales(model, tokenizer, dataset_path, num_samples=512, seq_len=512):
    model.eval()
```

```
    device =  next(model.parameters()).device

    # TODO: 使用 hook 函数统计激活值的最大绝对值
    # note: Hook 是一种在程序运行过程中“插入自定义逻辑”的机制，用于在特定事件（如前向、反向传播）发生时执行额外代码。

    act_scales = {} # torch.Size([2048])

    def stat_input_hook(m, x, y, name):
        # hook 函数
        raise NotImplementedError

    # hint: 向所有线性层注册 hook
    #       使用 model.named_modules() 函数遍历模型的 名称和模块
    #       输出形如 {String: torch.Size([2048])} 的层名和激活值，
    #       保存在 act_scales 字典中
    #       注册前向传播时候的 hook，可使用模块的
    #       register_forward_hook() 函数

    dataset = load_dataset("parquet", data_files=dataset_path,split="train")
    dataset = dataset.shuffle(seed=42)

    for i in tqdm( range(num_samples)):
        # text = "Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.\n\n### Instruction:\n{instruction}\n\n### Input:\n{input}\n\n### Response:\n".format(
            #       instruction=dataset["instruction"][i], input=dataset["output"][i]
            # )
        text = "Below is an instruction that describes a task, paired with an input that provides further context. "
        "Write a response that appropriately completes the request.\n\n"
        "### Instruction:\n"
        "Continue the following text naturally, maintaining the same style and coherence.\n\n"
        "### Input:\n"
        f"{dataset['text']}\n\n"
        "### Response:\n"
        input_ids = tokenizer(
```

```
            text, return_tensors="pt", max_length=seq_len, truncation=True
        ).input_ids.to(device)
        model(input_ids)

    # TODO: 删除 hooks, 你需要提前记录所有 hook 的句柄
    # hint: register_forward_hook() 函数的返回值即为 hook 的句柄

    return NotImplementedError() # 返回值为 act_scales 字典
```

进行 W8A8 量化

基于采集的激活最大值，对模型中线性层进行 W8A8 量化，替换模型中所有线性层为 `W8X8Linear`，即可获得量化后的 TinyLLaMA 模型。

```
class W8X8Linear(nn.Module):
    def __init__(self, ori_w:Tensor, bias: Optional[Tensor] = None,act_max:Optional[
        ↪ Tensor] = None,alpha=32):
         super().__init__()
        self.bias = None if bias is None else nn.Parameter(bias,requires_grad=False)
        self.dtype = ori_w.dtype
        self.alpha = alpha
        self.scales = None
        if act_max is not None:
            act_max = act_max.to(ori_w.device)
            self.scales = (act_max. pow(alpha) / ori_w. abs(). max(dim=0)[0].
                pow(1 - alpha)).clamp( min=1e-5).to(dtype=ori_w.dtype)
            self.scales = nn.Parameter(self.scales,requires_grad=False)
            ori_w = ori_w.detach().mul(self.scales)
        self.weight_q,self.max_val = quantize_mat(ori_w.detach())
        self.weight_q = nn.Parameter(self.weight_q.t(),requires_grad=False)
        self.max_val = nn.Parameter(self.max_val,requires_grad=False)

    def forward(self,x:Tensor) -> Tensor:
        # TODO
```

3.7.3 步骤三：导出 ONNX 模型

ONNX 是一种开放的神经网络中间表示标准，支持多种深度学习框架（如 PyTorch、TensorFlow、MindSpore 等）之间的模型转换与部署。在完成模型的权重

量化与激活值量化后，需要将模型导出为中间表示格式——ONNX（Open Neural Network Exchange）。

导出流程的主要步骤如下：

1. **加载模型与权重**：首先加载量化后的 TinyLLaMA 模型及其对应的 Tokenizer。模型在 CPU 上执行前向计算以确保导出过程的稳定性。同时导入量化配置文件（`quant_cfg.py`），其中包含每一层的量化比例与激活值缩放参数。
2. **构造导出输入样例（Dummy Input）**：由于 ONNX 需要在导出时执行一次前向推理，因此必须定义模型的输入张量。对于大语言模型，这通常包括：
   - `input_ids`: 模型输入的 token 序列；
   - `attention_mask`: 表示注意力范围的掩码；
   - `position_ids`: 序列中每个 token 的位置索引；
   - `past_key_values`: 用于缓存历史注意力状态以加速推理；

   这些输入的形状需严格与模型配置保持一致，否则可能导致 ONNX 导出失败或在编译阶段出现维度不匹配错误。
3. **定义动态轴（Dynamic Axes）**：为了支持不同的输入长度（如批次大小与上下文长度），ONNX 导出时需要显式声明动态维度。在实验中，定义了：
   - `batch_size`: 动态批大小；
   - `seq_length`: 当前输入序列长度；
   - `kv_len`: 已缓存的上下文长度；

   动态轴的定义有助于模型在推理时处理不同长度的输入，提高灵活性。
4. **执行导出**：使用 PyTorch 的 `torch.onnx.export()` 接口将模型结构与参数权重导出为 ONNX 文件。参数如下：
   - `opset_version=13`：ONNX 算子版本；
   - `export_params=True`：导出模型权重；
   - `input_names`、`output_names`：明确输入输出节点；
   - `dynamic_axes`：定义可变维度；

```python
def export_onnx(base_model,out_path,quant_cfg_path,act_path):
    tokenizer = LlamaTokenizer.from_pretrained(base_model)
    device = "cpu"
    model = LlamaForCausalLM.from_pretrained(
        base_model,
        torch_dtype=torch.float16,
        device_map="auto",
    ).to(device)
    model_cfg=model.model.config
    spec = importlib.util.spec_from_file_location("quant_cfg_module", quant_cfg_path)
    quant_cfg_module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(quant_cfg_module)
    quantize_cfg = quant_cfg_module.get(model_cfg,act_path)
    from quantize import quantize
    quantize(model,quantize_cfg)

    input_names = ["input_ids", "attention_mask", "position_ids","past_key_values"]
    output_names = ["logits","out_key_values","attn_scores"]

    batch_size,seq_len,kv_len=1,1,1024 # 请勿修改

    # TODO: 构造导出假输入，需要和模型实际输入对应上
    # input_ids 形状为 (batch_size, sequence_length)
    # attention_mask 形状为 (batch_size, all_sequence_length(what's that?))
    # position_ids 形状为 (batch_size, sequence_length)
    # past_key_values 形状为 (n_layers, 2, batch_size, n_heads, kv_len, head_dim)

    input_args = (
        # TODO: 填写下面的输入
        # input_ids: torch.LongTensor = None,
        # attention_mask: Optional[torch.Tensor] = None,
        # position_ids: Optional[torch.LongTensor] = None,
        # past_key_values: Optional[List[torch.FloatTensor]] = None,
        # inputs_embeds: Optional[torch.FloatTensor] = None,          # 这里填 None
        # labels: Optional[torch.LongTensor] = None,                  # 这里填 None
        # use_cache: Optional[bool] = None,                           # 这里填 True 才
            ↪ 能输出 past_key_values
        # output_attentions: Optional[bool] = None,                   # 这里填 True
    )
```

```
39
40      dynamic_axes = {
41          # TODO: 根据上面构造的输入输出，填写 dynamic_axes
42          # 格式为：
43          # name: { axis_index: "axis_name", ... }
44          # 如：
45          # "input_ids": { 0: "batch_size", 1: "seq_length" }
46          #
47          # hint: 格外注意 past_key_values，哪些维度是可变的？
48      }
49
50
51      model. eval()
52      torch.onnx.export(
53          model,
54          f=out_path,
55          args=input_args,
56          input_names=input_names,
57          output_names=output_names,
58          dynamic_axes=dynamic_axes,
59          opset_version=13,
60          export_params=True,
61      )
```

3.7.4 步骤四：使用 ATC 编译生成 OM 模型

使用昇腾提供的 `atc` 工具将 ONNX 模型编译为可在 NPU 上运行的 OM 模型：

```
1 atc --framework=5 --model="./TinyLlama-chat-v1.0-quant-ascend.onnx"
    ↪ --output="./TinyLlama-chat-v1.0-quant" --input_format=ND --
    ↪ input_shape="input_ids:1,1;attention_mask:1,1025;position_ids:1,1;
    ↪ past_key_values:22,2,1,4,1024,64" --log=debug --soc_version=
    ↪ Ascend310B4 --precision_mode=must_keep_origin_dtype
```

其中，model 参数为 onnx 模型路径，output 为输出路径，其他如无问题请勿改动

3.7.5 步骤五：推理与输出生成

聊天模板

实现聊天模板，prompt 有三种类型的角色，请参考下面的格式进行实现，输出为

格式参考: https://huggingface.co/docs/transformers/main/en/chat_templating

输入样例:

```
messages = [
    {
        ”role”: ”system”,
        ”content”: ”You are a friendly chatbot who always responds in the
            ↪ style of a pirate”,
    },
    {   ”role”: ”user”,
        ”content”: ”How many helicopters can a human eat in one sitting?”
    },
    {   ”role”: ”assistant”,
        ”content”: ”I do not know!”
    },
]
```

输出样例:

```
<|system|> # 注意这里到下一行需要使用换行符 \n 进行换行
You are a friendly chatbot who always responds in the style of a pirate</s>
<|user|>
How many helicopters can a human eat in one sitting?</s>
<|assistant|>
I do not know!</s>
```

注意，如果最后一条消息是用户消息，则生成的文本以 <|assistant|> 结尾，表示模型需要生成回复

```python
def apply_chat_template(self,messages:List[Dict[ str, str]]) ->  str:
    # TODO
    return NotImplementedError()
```

logits 采样与生成策略

在自回归生成任务中，模型在每一步返回一个 logits 向量（长度为词表大小），表示下一个 token 的未归一化对数概率。将 logits 转换为实际采样分布并选择下一 token 的过程称为采样策略（sampling strategy）。下面说明 Top-k 并给出实现要点与数值稳定性技巧。

数值稳定 softmax 给定 logits 向量 $\mathbf{z} \in \mathbb{R}^V$（$V$ 为词表大小），softmax 的直接计算为

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

为避免溢出/下溢，应首先做数值偏移（减去最大值）：

$$\tilde{z}_i = z_i - \max_j z_j, \quad p_i = \frac{e^{\tilde{z}_i}}{\sum_j e^{\tilde{z}_j}}.$$

若要使用温度（temperature）参数 $T > 0$，在 softmax 前放缩 logits：

$$\tilde{z}_i = \frac{z_i}{T} - \max_j \frac{z_j}{T}.$$

Top-k 采样 Top-k 将 logits 的采样空间限制在概率最高的 k 个 token 内：

1. 找到前 k 个 logits 的索引与对应 logits。
2. 对这 k 个 logits 做 softmax（或在采样前做温度缩放）。
3. 在该子分布上进行加权随机采样。

```python
def sample_logits_top_k(
    self,
    logits: np.ndarray,
    sampling_value:  float = None,
    temperature:  float = 1.0,
```

```
) -> np.ndarray:
    logits = logits.astype(np.float32)
    # TODO: 只在概率最高的前 k 个 tokens 中加权采样

    return NotImplementedError() # 输出为一个 numpy 数组，表示采样得到的下一个 token id
```

推理主循环与输出解码

自回归推理主循环通常如下：

1. 使用 `apply_chat_template` 将消息格式化为 prompt 文本；
2. 使用 tokenizer 将 prompt 编码为 `input_ids`；
3. 将 `input_ids` 和 `past_key_values` 输入到模型；
4. 模型返回 `logits` 和更新后的 `past_key_values` 以及 `attention_score`（这个参数在实验中不会使用到）；
5. 从最后一个位置的 logits 中采样得到 next_token（使用上节策略）；
6. 将 next_token 追加到生成序列，并更新 attention_mask、position_ids 以及传入下一步的 input_ids（通常为单个 token）；
7. 检查是否遇到 EOS 或满足停止条件，如是则结束，并回退 kv；否则回到第 3 步循环直至达到最大长度。

```
def predict(self, text):
    if text == "":
        return
    self.format_last_output()
    # TODO
    return NotImplementedError()
```

3.7.6 代码目录

```
.
+-- export_llama
|    +-- config
|    |    +-- w8.py
```

```
|   |   +-- w8x8.py
|   +-- export_llama.py
|   +-- generate_act_scales.py
|   +-- quantize.py
+-- inference
    +-- config.py
    +-- engine.py
    +-- inference.py
    +-- kvcache.py
    +-- main.py
    +-- session.py
```

3.8 评分指标

模型、分词器相关文件不需要上传，仅需要代码。

评分指标分为两个部分，其中实验代码和结果占 80%，实验报告的撰写（包含实验思考部分）占 20%。

实验结果的评分指标如下：

1-59：代码无法跑通，将进行代码审计人工评分。

60：代码正确运行，模型成功加载，推理时能够正常输出合乎逻辑的语句。

80: 评估推理的执行效率，尝试不同的实现方式能否带来效率的提升。

实验报告的评分指标如下：

实验报告撰写计 10%，实验思考计 10%。

选做视情况会有额外加分，但总分不会超过 100 分。

3.9 实验思考

1、模型推理输出得到的 logits 的是什么数据类型、什么形状的张量？这个形状和模型论文中的 vocab_size (=32000) 有什么关系？

hint: 使用 logits.dtype, logits.shape

2、在遇到哪些特殊 token 时，需要回退 KV Cache 以防止推理意外停止？如果不回退，会发生什么？

3、简述你在实验中是如何从一个只能进行“下一个单词预测”的模型，到实现一个能实现相互对话的“AI”。

hint: apply_chat_template()

4、量化（Quantiztion）是什么？权重量化和激活值量化，在实现上有什么区别？不同量化方式（如 W8A8、W4A8）对模型推理效率和精度有何影响？

5、你是否能基于实验中部署的大模型开发其他有趣的应用？（选做）