|  |
| --- |
| 基于流水线并行和卸载策略的大模型精调系统 V1.0  软件说明书 |
|  |

**目录**

[1 简介 3](#_Toc3563)

[1.1 编写目的 3](#_Toc30505)

[1.2 使用对象 3](#_Toc23018)

[2 系统需求 3](#_Toc13284)

[3 软件流程概述 3](#_Toc17968)

[3.1 总体流程 3](#_Toc31186)

[3.2 训练流程 4](#_Toc32206)

[4 安装和运行方法 6](#_Toc14820)

[4.1 获取软件源代码 6](#_Toc32624)

[4.2 依赖环境准备 6](#_Toc2965)

[4.3 配置和使用 7](#_Toc12856)

[4.4 命令行参数说明 7](#_Toc30918)

[4.5 程序执行结果 8](#_Toc25257)

[5 软件运行截图 8](#_Toc25995)

[5.1 显示帮助内容 8](#_Toc28596)

[5.2 运行核心功能 9](#_Toc23416)

# 产品概述

本软件提供了一个大模型精调框架，优化大模型精调的显存占用及设备间通信效率，提升性能与资源利用率。旨在通过高效的流水线并行和模型卸载策略，帮助用户在有限的硬件条件下实现大模型的精调任务。软件可以帮助用户实现的功能表现为从开源网站引用大模型，使用开源数据集，在单节点上高效利用多设备资源并行精调大模型。在保证模型精度和训练效率的前提下，本技术将可精调的模型规模提升至单设备方法的**5倍**。例如，在传统单设备精调方法仅支持 **1B 参数规模**的情况下，本技术可扩展至 **5B 参数规模**的大模型精调，突破单设备显存瓶颈，实现更大规模模型的高效训练。

## 编写目的

本文档为使用说明文档，为产品的使用与维护提供信息基础。

## 使用对象

本文档的使用对象主要为产品的使用人员。产品的目标用户是**有精调大模型需求但硬件资源有限的用户群体。**

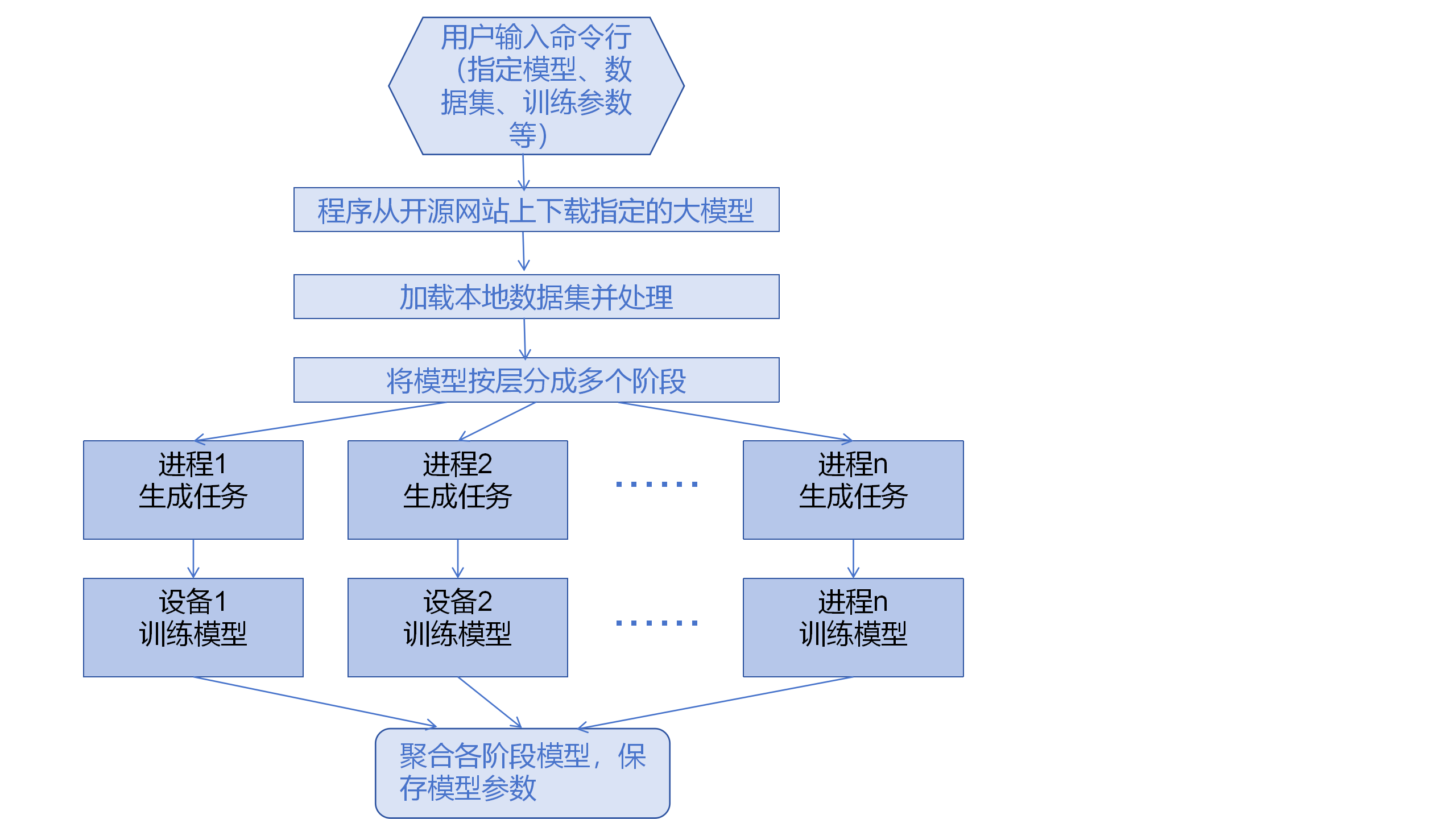
# 环境需求

本软件适用系统：Linux系统。推荐Ubuntu 22.04.5 LTS操作系统。

# 软件流程概述

## 总体流程

基于流水线并行和卸载策略的大模型精调系统总体流程如图3-1总体流程图所示。



**图 3-1 总体流程图**

## 训练流程

**3.2.1背景和目的**

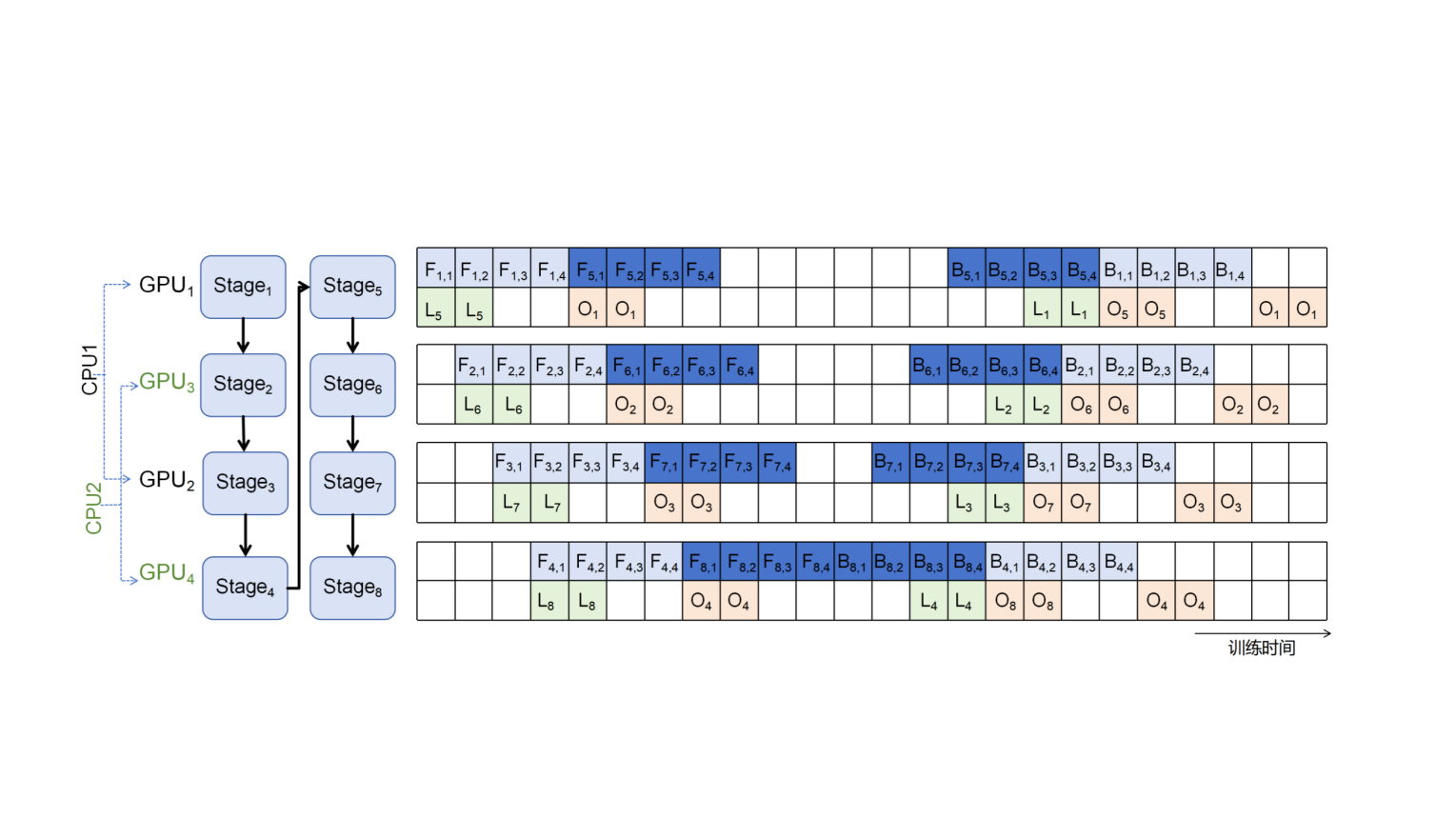
本流程图展示了在特定模型和硬件设备条件下，采用流水线并行技术和模型卸载/加载技术进行模型精调（Fine-tuning）的训练过程。通过将模型划分为多个阶段，并利用多个加速设备并行处理不同阶段的任务，本系统显著提高了训练效率。通过在训练过程中卸载/加载模型参数，本系统显著提高了显存利用率。

**3.2.2模型精调的一般过程**

模型精调（Fine-tuning）是指在预训练模型的基础上，通过进一步训练使其适应特定任务的过程。通常，模型精调包括以下步骤：

* **加载预训练模型**：从预训练模型中加载权重和参数。
* **划分数据集**：将训练数据划分为多个微批次（Mini-batches）。
* **前向传播**：计算每个微批次的输出。
* **计算损失**：根据输出结果和真实标签计算损失值。
* **反向传播**：根据损失值更新模型参数。
* **迭代训练**：重复上述步骤，直到模型收敛。

**3.2.3训练流程图以及内容详细说明**



**图 3-2 训练流水线示意图**

* 模型划分：将模型划分为 8 个阶段，每个阶段由不同的加速设备负责。例如：

- 设备 1 负责阶段 1 和阶段 5。

- 设备 2 负责阶段 2 和阶段 6。

- 以此类推。

* 操作符号

- Fi,j：表示对阶段 i 的第 j 个微批次进行前向传播操作。

- Bi,j：表示对阶段 i 的第 j 个微批次进行反向传播操作。

- Li：表示对阶段 i 的模型参数进行加载操作。

- Oi：表示对阶段 i 的模型参数进行卸载操作。

* 流水线并行

- 每个加速设备同时处理不同微批次的不同阶段任务。例如：

设备 1 在处理阶段 1 的第 j 个微批次的同时，设备 2 已经开始处理阶段 2 的第 j 个微批次。

- 通过这种流水线并行的方式，系统能够充分利用硬件资源，减少空闲时间，提高训练效率。

* 动态模型移动

在前向传播/反向传播计算前，将模型参数加载至加速设备，传播计算完成后，将模型参数卸载至CPU。这种方法能够提高显存利用率，为中间结果提供更多可用的显存。

**3.2.4总结技术优势**

本系统的创新点在于：

* 单设备多阶段流水线并行：传统流水线并行将模型层分割为N个阶段，其中N表示GPU数量。我们采用虚拟流水线策略，将模型层分割为v×N个阶段，每个GPU轮流执行v个流水线阶段。以N=4，v=2为例。8个虚拟阶段（v×N=2×4）在4个GPU间循环分布：GPU1处理阶段1和5，GPU2处理阶段2和6，GPU3处理阶段3和7，GPU4处理阶段4和8。动态加载和卸载：通过动态加载（Li）和卸载（Oi）模型参数，减少了设备间的通信开销。
* 模型卸载：在训练过程中动态将模型参数卸载到CPU，为激活值和梯度节省高带宽内存（HBM）。
* 通信-计算重叠：优化流水线执行，使得模型移动带来的通信和计算操作尽量重叠，减少设备计算资源空闲时间。
* 交叉映射：战略性地放置流水线阶段以最小化通信开销。
* 多流多线程执行：并发数据移动和计算以提高吞吐量。

# 安装和运行方法

## 获取软件源代码

用户需从 GitHub 获取本软件的源代码，可使用以下命令下载：

git clone [https://github.com/MLSysU/Mobius/commits/bishe/](https://github.com/MLSysU/Mobius/commits/tmp/)

cd Gpipe

## 

## 依赖环境准备

本软件支持在不同硬件环境上运行，用户需根据自身的硬件选择合适的依赖环境。  
推荐使用 conda 创建虚拟环境，安装必要的依赖项：

conda create -n finetune-env python=3.9

conda activate finetune-env

然后安装依赖：

pip install -r requirements.txt

如果使用 GPU，请确保已正确安装 CUDA 和 cuDNN，并安装适配的 PyTorch 版本。

或者，可以直接使用docker镜像：

docker pull coir1hat1man/mobius:latest

## 配置和使用

在运行软件前，用户需要准备数据集和开源模型地址，并指定相关参数。可以通过命令行运行（举例）：

下方命令解释：如果使用GPU，CUDA\_VISIBLE\_DEVICES用于指定GPU的编号；使用torchrun运行程序；nproc\_per\_node用于指定单节点上使用的设备数；master\_port用于指定运行程序的端口；main.py是程序的主文件；其他程序命令行参数说明见表4-1。

CUDA\_VISIBLE\_DEVICES=0,1,2,3 torchrun --nproc\_per\_node 4 --master\_port 29502 ./main.py --model\_path=’meta-llama/Llama-2-7b-hf’

--dataset=’xsum’

--num\_iterations=20 --batch\_size=64 --num\_stage=8 --use\_prefetch --use\_offload

## 

## 命令行参数说明

**表 4-1命令行参数表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **参数名** | **参数类型** | **默认值** | **参数描述** |
| 1 | model\_path | 字符串 | 'meta-llama/Llama-2-7b-hf' | 开源模型名称 |
| 2 | dataset | 字符串 | ‘xsum’ | 本地数据集名称 |
| 3 | save\_params | 字符串 | 'finetune\_params.pt' | 精调完成之后，保存模型新参数的文件名 |
| 4 | use\_prefetch | 布尔 |  | 使用prefetch技巧 |
| 5 | no\_prefetch | 布尔 |  | 不使用prefetch技巧 |
| 6 | use\_offload | 布尔 |  | 使用模型参数卸载技巧 |
| 7 | no\_offload | 布尔 |  | 不使用模型参数卸载技巧 |
| 8 | batch\_size | 整数 | 128 | 批次大小 |
| 9 | num\_chunks | 整数 | 4 | 微批次数 |
| 10 | seq\_length | 整数 | 512 | 句长 |
| 11 | embedding\_dim | 整数 | 4096 | 嵌入层维度 |
| 12 | ff\_dim | 整数 | 4096 | 前馈层维度 |
| 13 | num\_iterations | 整数 | 2 | 迭代次数 |
| 14 | num\_stages | 整数 | 8 | 将模型切分成的阶段数 |
| 15 | num\_layers | 整数 | 8 | 训练的模型层数 |
| 16 | num\_heads | 整数 | 32 | 多注意力头数 |

## 程序执行结果

在精调完成后，将优化后的模型参数保存至指定的 .pt 文件中，以便用户在后续推理或其他下游任务中高效加载并使用该模型。

## 常见故障排查

**问题 1**：安装 PyTorch 时出现版本冲突

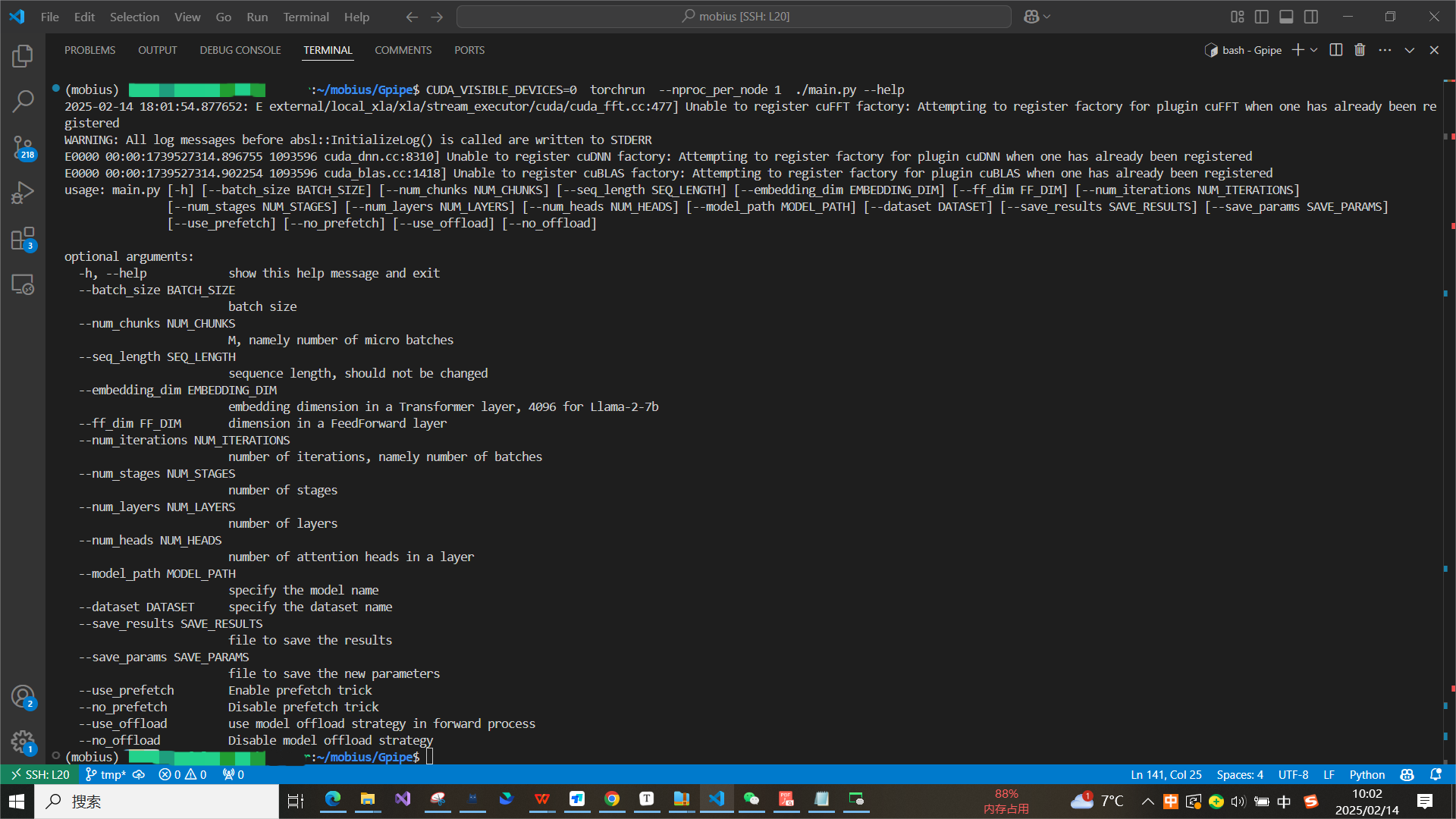
**解决方法**：确保 CUDA 和 PyTorch 版本匹配。[参考 PyTorch 官方安装指南](https://pytorch.org/get-started/previous-versions/)。

**问题 2：**运行时报错 "NCCL 未初始化"

**解决方法：**检查环境变量 LOCAL\_RANK 和 WORLD\_SIZE 是否已正确设置。

# 软件运行截图

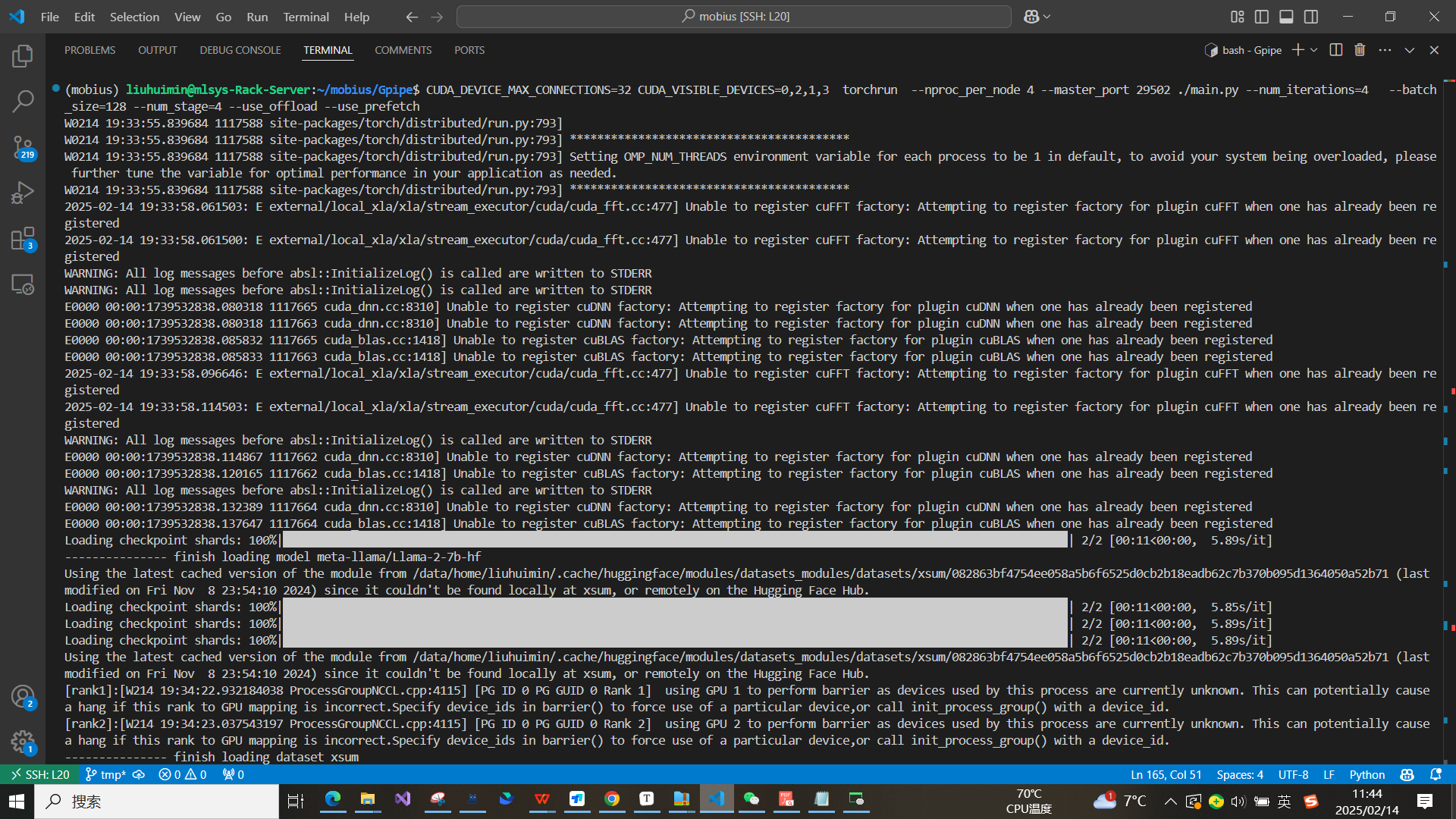
## 显示帮助内容



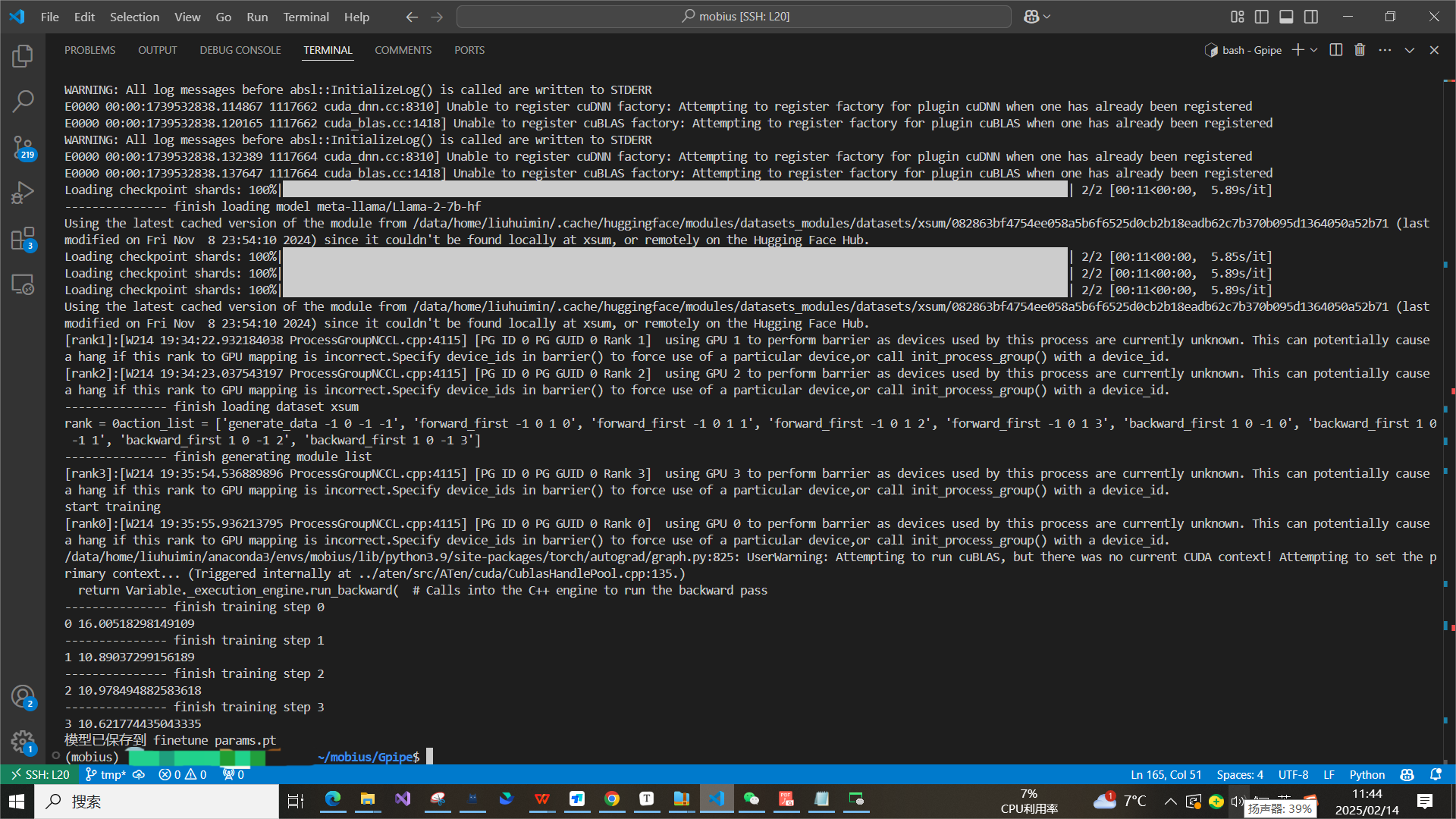
**图 5-1 显示帮助运行图**

## 运行核心功能

执行运行指令后，根据终端提示语，可以看到程序依次完成模型下载、加载数据集、切分模型、迭代训练模型。最终精调完成的模型参数以.pt文件的形式存储在本地。

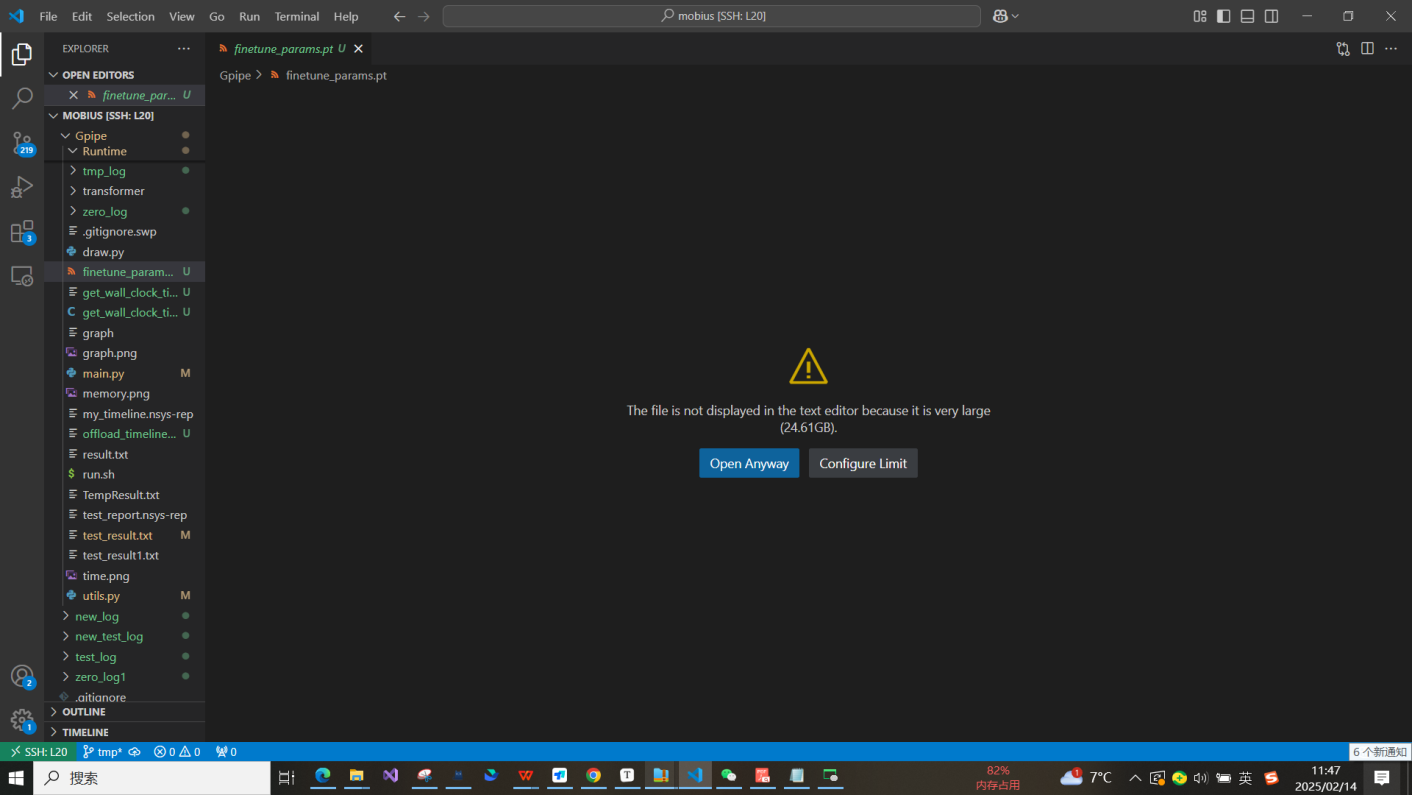


**图 5-2-a 核心功能运行图**



**图 5-2-b(接5-2-a) 核心功能运行图**

最终生成的模型参数文件：



**图 5-3 程序运行结果图**