

CANN 软件安装指南

文档版本

01

发布日期

2024-10-15



华为技术有限公司



版权所有 © 华为技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI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。

目 录

1 安装须知..... 1

2 安装方案..... 3

2.1 非昇腾设备..... 3

2.2 Atlas 200（EP 场景）..... 4

2.3 Atlas 200（RC 场景）..... 5

2.4 Atlas 200I A2（EP 场景）..... 7

2.5 Atlas 200I A2（RC 场景）..... 8

2.6 Atlas 200I SoC A1..... 9

2.7 A300-3000、A300-3010..... 10

2.8 Atlas 300I Pro、Atlas 300V Pro、Atlas 300V、Atlas 300I Duo..... 11

2.9 Atlas 500 A2..... 13

2.10 A500 Pro-3000..... 13

2.11 A800-3000、A800-3010..... 15

2.12 A300T-9000、Atlas 300T Pro、A800-9000、A800-9010、A900-9000..... 16

2.13 Atlas 200T A2 Box16、Atlas 300T A2、Atlas 800T A2、Atlas 900 A2 PoD..... 18

3 准备硬件环境..... 21

4 安装驱动和固件..... 25

5 CANN 软件包支持的操作系统..... 27

6 安装开发环境..... 32

6.1 准备软件包..... 32

6.2 准备安装及运行用户..... 33

6.3 安装依赖..... 34

6.3.1 安装前必读..... 34

6.3.2 依赖列表..... 34

6.3.3 安装步骤（Ubuntu 22.04）..... 38

6.3.4 安装步骤（openEuler 22.03）..... 41

6.3.5 安装步骤（SLES 12.5）..... 43

6.4 在非昇腾设备上安装..... 46

6.4.1 安装开发套件包..... 46

6.4.2 安装二进制算子包..... 47

6.4.3 配置环境变量..... 48

6.4.4 配置交叉编译环境..... 48

6.5 在昇腾设备上安装..... 49

6.5.1 安装开发套件包..... 49

6.5.2 安装二进制算子包..... 50

6.5.3 配置环境变量..... 51

6.5.4 安装深度学习框架..... 51

6.5.4.1 安装 TensorFlow..... 51

6.5.4.1.1 安装开源框架 TensorFlow..... 51

6.5.4.1.2 安装框架插件包..... 54

6.5.4.2 安装 PyTorch..... 55

6.5.4.2.1 安装前必读..... 55

6.5.4.2.2 安装 PyTorch 框架..... 56

6.5.4.2.3 安装 torch_npu 插件..... 59

6.5.4.2.4 安装 APEX 模块..... 60

6.5.4.2.5 安全说明..... 61

6.5.4.3 安装昇思 MindSpore..... 68

6.5.5 安装 Python 版本的 proto..... 68

6.5.6 配置 device 的网卡 IP..... 69

7 安装运行环境（nnrt 软件，在物理机/虚拟机安装）..... 71

7.1 安装前必读..... 71

7.2 准备软件包..... 72

7.3 准备安装及运行用户..... 73

7.4 安装离线推理引擎包..... 74

7.5 配置环境变量..... 75

8 安装运行环境（nnae 软件，在物理机/虚拟机安装）..... 77

8.1 安装前必读..... 77

8.2 准备软件包..... 78

8.3 准备安装及运行用户..... 79

8.4 安装依赖..... 80

8.4.1 安装前必读..... 80

8.4.2 依赖列表..... 80

8.4.3 安装步骤（Ubuntu 22.04）..... 84

8.4.4 安装步骤（openEuler 22.03）..... 86

8.4.5 安装步骤（SLES 12.5）..... 89

8.5 安装深度学习引擎包..... 92

8.6 安装二进制算子包..... 93

8.7 配置环境变量..... 94

8.8 安装深度学习框架..... 94

8.8.1 安装 TensorFlow..... 94

8.8.1.1 安装开源框架 TensorFlow..... 94

8.8.1.2 安装框架插件包..... 97

8.8.2 安装 PyTorch..... 98

8.8.2.1 安装前必读.....	98
8.8.2.2 安装 PyTorch 框架.....	100
8.8.2.3 安装 torch_npu 插件.....	102
8.8.2.4 安装 APEX 模块.....	103
8.8.2.5 安全说明.....	104
8.8.3 安装昇思 MindSpore.....	111
8.9 安装 Python 版本的 proto.....	111
8.10 配置 device 的网卡 IP.....	113
9 升级.....	114
9.1 升级前必读.....	114
9.2 升级操作.....	115
10 卸载.....	118
11 后续任务.....	120
12 安全加固.....	121
12.1 加固须知.....	121
12.2 Host 侧加固.....	121
12.2.1 禁止使用 root 账户远程登录系统.....	121
12.2.2 账户 UID 安全加固.....	121
12.2.3 内存地址随机化机制安全加固.....	121
12.2.4 禁止使用 SetUID 或 SetGID 的 shell 脚本.....	122
12.2.5 无属主文件安全加固.....	122
12.2.6 设置 umask.....	122
A 常用操作.....	123
A.1 安装、回退和卸载 CANN 补丁包.....	123
A.2 编译安装 Python3.7.5.....	125
A.3 安装 3.5.2 版本 cmake.....	125
A.4 安装 7.3.0 版本 gcc.....	126
A.5 配置网卡 IP 地址.....	127
A.6 设置用户有效期.....	128
A.7 配置 pip 源.....	129
A.8 查询软件包版本信息.....	130
B FAQ.....	131
B.1 安装驱动时提示 “The user and group are not same with last installation”	131
B.2 openssl-devel 安装时报错提示 so 文件冲突.....	131
B.3 pip3 安装软件包时，出现 read time out 报错.....	132
B.4 pip3 install scipy 报错.....	133
B.5 pip3 install numpy 报错.....	134
B.6 pip3 install 报错 “subprocess.CalledProcessError: Command '('lsb_release', '-a')' return non-zero exit status 1”	135
B.7 Python 版本不一致导致的问题.....	136

B.8 安装 run 包失败，日志中报错 “usergroup=root not right! ” 137

B.9 使用 glibc 2.28 版本运行业务调用 malloc 接口出现 Core Dump 问题..... 137

B.10 openEuler 22.03 上运行业务时，出现 firewalld 相关 soft lockup 或 ksoftirqd 占用 CPU 过高..... 138

B.11 安装原生 PyTorch 框架时使用华为源下载 typing 依赖导致 Python 环境错误..... 138

B.12 PyTorch 框架在 ARM CPU 上算子计算结果异常..... 139

B.13 原生 PyTorch 编译过程报错 no module named yaml/typing_extensions..... 140

B.14 PyTorch 运行遇到找不到 te 问题..... 140

B.15 PyTorch 编译过程提示 CMAKE 相关错误..... 141

B.16 PyTorch 运行提示找不到 libblas.so..... 141

B.17 PyTorch 在容器中运行未挂载 device 问题..... 142

B.18 PyTorch 运行 import torch_npu 显示_has_compatible_shallow_copy_type 重复注册 warning 问题.... 143

B.19 PyTorch 运行 import torch_npu 显示 No module named 'torch_npu_C'..... 143

B.20 PyTorch 编译时出现 Breakpad error: field 'regs' has incomplete type
'google_breakpad::user_regs_struct'报错..... 144

C 附录..... 145

C.1 参数说明..... 145

C.2 相关信息记录路径..... 148

C.3 AICPU Kernel 加载说明..... 149

C.4 口令复杂度要求..... 149

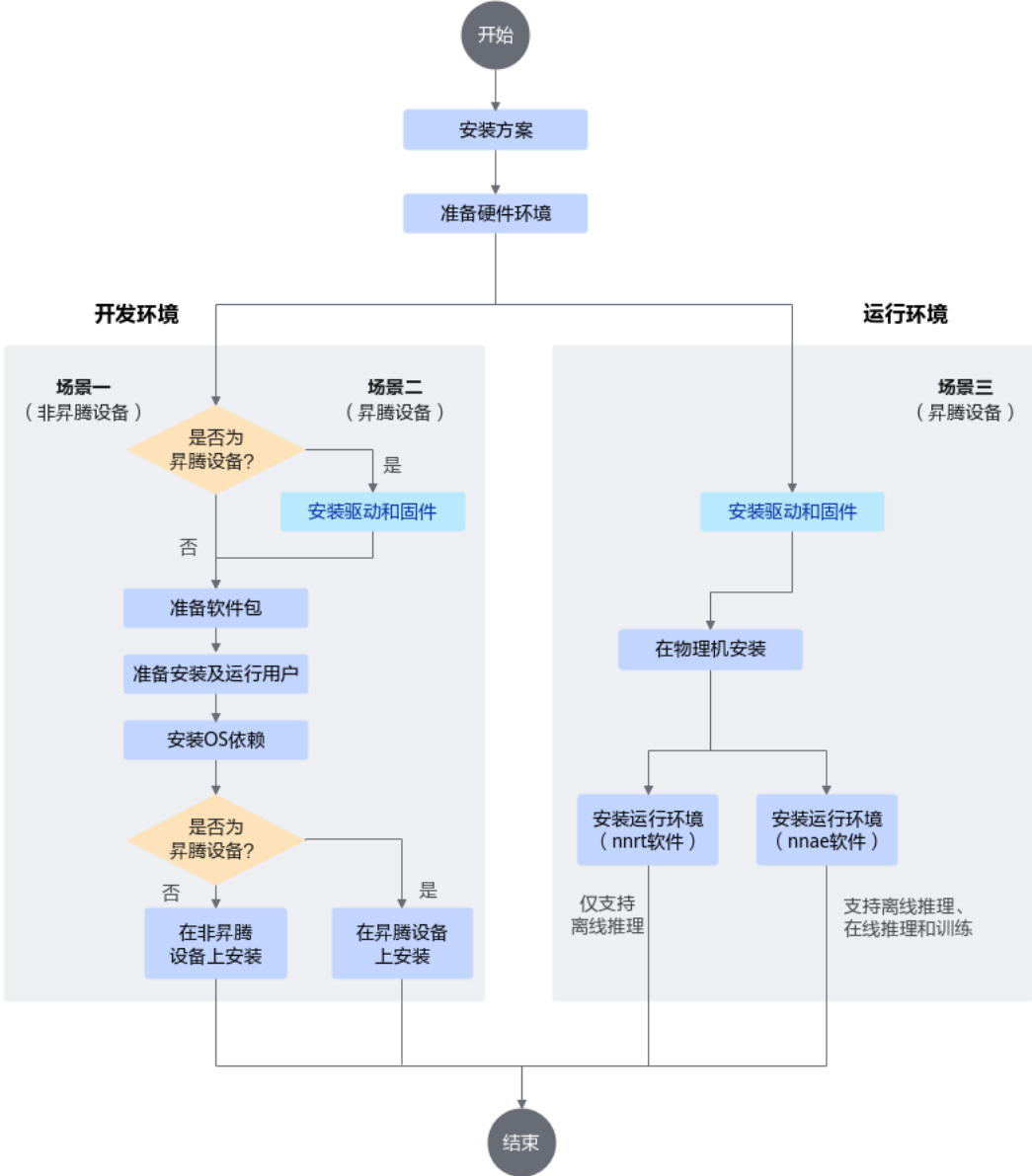
C.5 公网 URL..... 150

1 安装须知

CANN (Compute Architecture for Neural Networks) 是华为公司针对AI场景推出的异构计算架构，通过提供多层次的编程接口，支持用户快速构建基于昇腾平台的AI应用和业务。

本文档主要用于指导用户安装CANN开发或运行环境，文档中包含的硬件产品和PCIe工作模式的介绍具体请参考《昇腾产品形态说明》。

- **开发环境：**主要用于代码开发、编译、调测等开发活动。
 - （场景一）在非昇腾AI设备上安装开发环境，仅能用于代码开发、编译等不依赖于昇腾设备的开发活动（例如ATC模型转换、算子和推理应用程序的纯代码开发）。
 - （场景二）在昇腾AI设备上安装开发环境，支持代码开发和编译，同时可以运行应用程序或进行训练脚本的迁移、开发&调试。
- **运行环境：**在昇腾AI设备上运行用户开发的应用程序或进行训练脚本的迁移、开发&调试。



2 安装方案

2.1 非昇腾设备

2.2 Atlas 200（EP场景）

2.3 Atlas 200（RC场景）

2.4 Atlas 200I A2（EP场景）

2.5 Atlas 200I A2（RC场景）

2.6 Atlas 200I SoC A1

2.7 A300-3000、A300-3010

2.8 Atlas 300I Pro、Atlas 300V Pro、Atlas 300V、Atlas 300I Duo

2.9 Atlas 500 A2

2.10 A500 Pro-3000

2.11 A800-3000、A800-3010

2.12 A300T-9000、Atlas 300T Pro、A800-9000、A800-9010、A900-9000

2.13 Atlas 200T A2 Box16、Atlas 300T A2、Atlas 800T A2、Atlas 900 A2 PoD

2.1 非昇腾设备

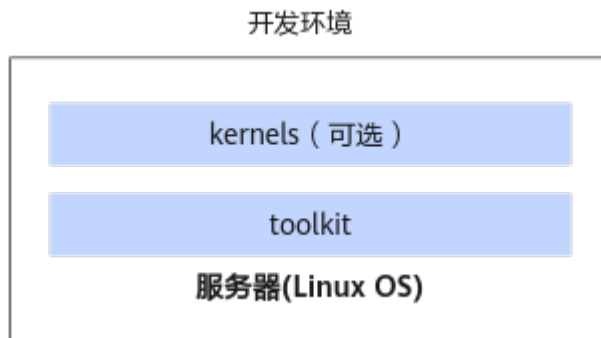
对于非昇腾AI设备，仅支持安装纯开发环境，安装软件如[图2-1](#)所示。

非昇腾AI设备无需安装固件与驱动，仅能用于代码开发、编译等不依赖于昇腾设备的开发活动，对于运行应用等需要依赖昇腾AI处理器的阶段，可将代码或编译后的可执行文件上传至昇腾AI处理器所在环境进行验证。

软件包说明：

- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的库文件和开发辅助工具等。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

图 2-1 开发环境



说明

若开发环境为x86_64架构，运行环境为aarch64架构，开发环境上可以只部署x86_64开发套件包，后续编译应用时需要使用交叉编译工具链及aarch64架构的库文件，可以参见[6.4.4 配置交叉编译环境](#)进行配置。

2.2 Atlas 200（EP 场景）

Atlas 200（Atlas 200 AI加速模块）以Atlas 200/300/500 推理产品的PCle（Peripheral Component Interconnect Express）的工作模式进行区分，如果PCle工作在从模式，则称为EP（Endpoint）模式；如果PCle工作在主模式，可以扩展外设，则称为RC（Root Complex）模式。具体可参考《昇腾产品形态说明》。

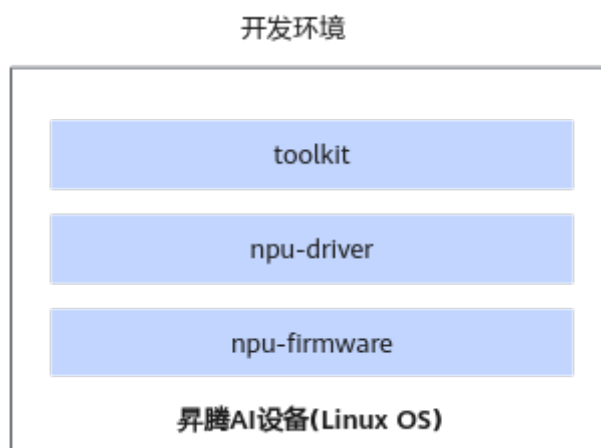
- 开发环境

开发环境安装软件如[图2-2](#)所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。

图 2-2 开发环境



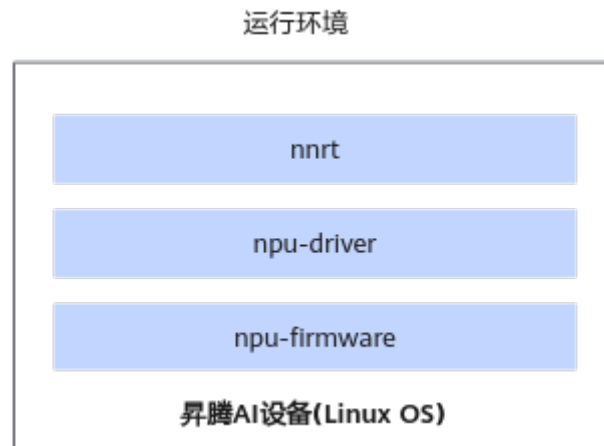
- 运行环境

运行环境安装软件如图2-3所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-3 运行环境



📖 说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.3 Atlas 200（RC 场景）

Atlas 200（Atlas 200 AI加速模块）以Atlas 200/300/500 推理产品的PCIe（Peripheral Component Interconnect Express）的工作模式进行区分，如果PCIe工作在主模式，可以扩展外设，则称为RC（Root Complex）模式；如果PCIe工作在从模式，则称为EP（Endpoint）模式。具体可参考《昇腾产品形态说明》。

- 开发环境

安装示意图如图2-4所示。

请先参考《Atlas 200 AI加速模块 6.0.0 软件安装与维护指南（RC场景）》制作并安装好Atlas 200 AI加速模块的启动镜像包后，再参考本文档安装toolkit软件包。

软件包说明：

toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。

图 2-4 开发环境



- 运行环境

安装示意图如图2-5所示。

Atlas 200 AI加速模块在容器内部署运行环境具体请参考《[Atlas 200 AI加速模块 6.0.0 软件安装与维护指南 \(RC场景\)](#)》。

软件包说明：

nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-5 运行环境



说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.4 Atlas 200I A2（EP 场景）

Atlas 200I A2（Atlas 200I A2 加速模块A200I A2 加速模块）以Atlas 200I/500 A2推理产品的PCIe（Peripheral Component Interconnect Express）的工作模式进行区分，如果PCIe工作在从模式，则称为EP（Endpoint）模式；如果PCIe工作在主模式，可以扩展外设，则称为RC（Root Complex）模式。具体可参考《昇腾产品形态说明》。

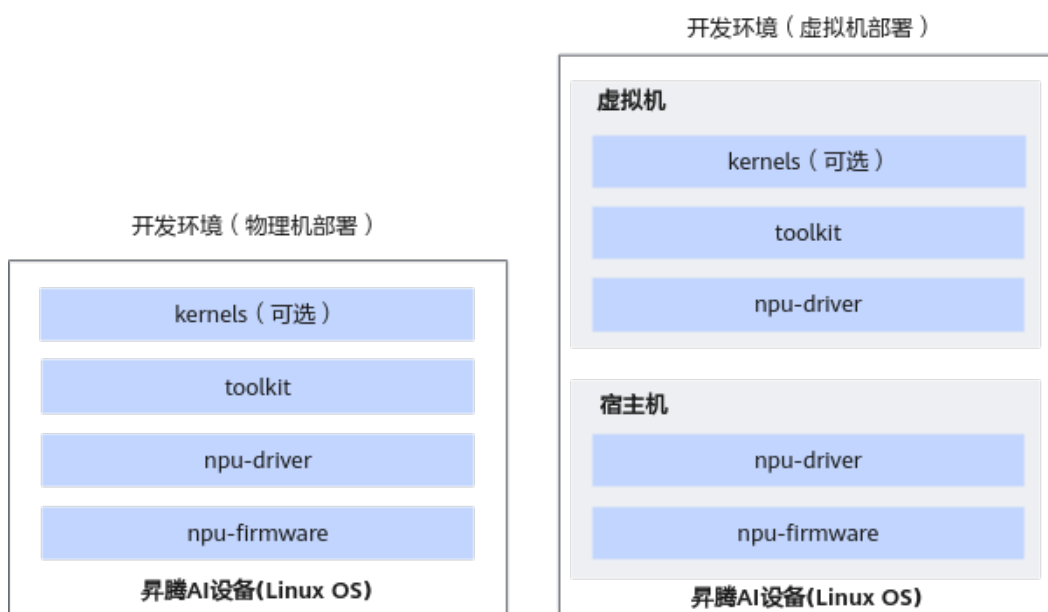
- 开发环境

开发环境安装软件如图2-6所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

图 2-6 开发环境



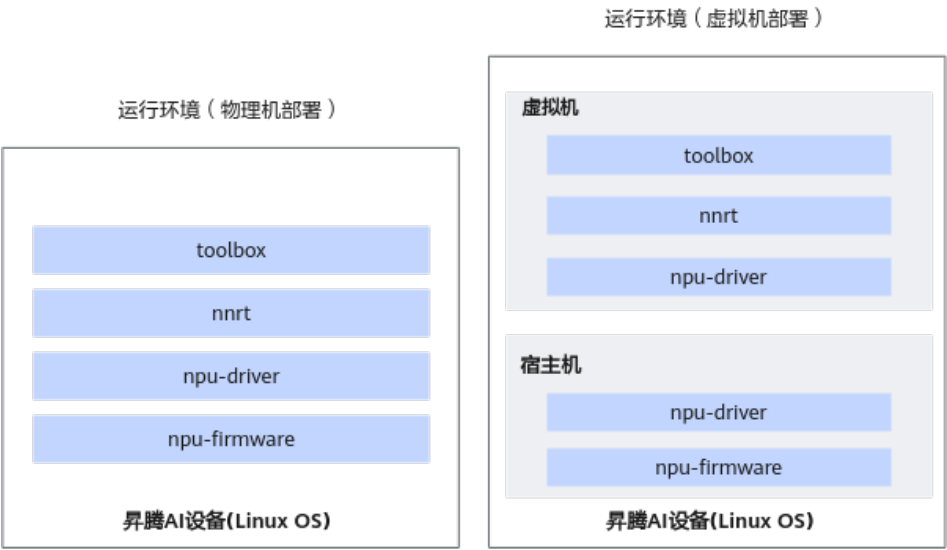
- 运行环境

运行环境安装软件如图2-7所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-7 运行环境



说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.5 Atlas 200I A2（RC 场景）

Atlas 200I A2（Atlas 200I A2 加速模块A200I A2 加速模块）以Atlas 200I/500 A2推理产品的PCIe（Peripheral Component Interconnect Express）的工作模式进行区分，如果PCIe工作在从模式，则称为EP（Endpoint）模式；如果PCIe工作在主模式，可以扩展外设，则称为RC（Root Complex）模式。具体可参考《昇腾产品形态说明》。

Atlas 200I A2（RC场景）仅支持作为运行环境。

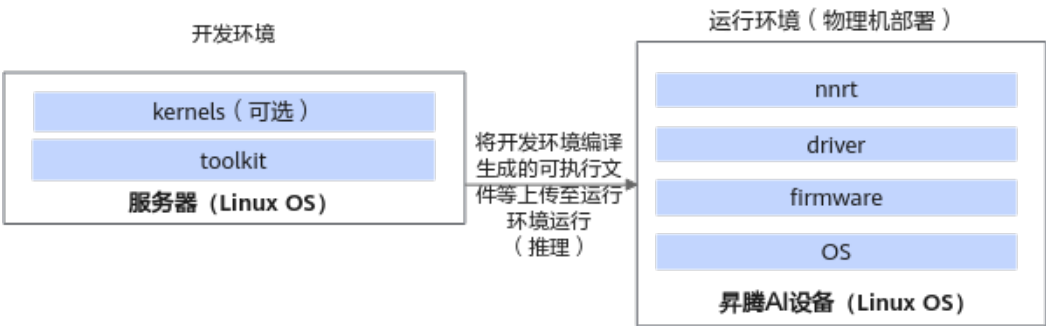
- 开发环境
需要另外准备一台服务器用于搭建开发环境，详情参考[2.1 非昇腾设备](#)。
- 运行环境
安装示意图如[图2-8](#)所示。

Atlas 200 AI加速模块（RC场景）在容器内部署运行环境具体请参考《[Atlas 200I A2 加速模块 23.0.0 昇腾软件安装指南](#)》。

软件包说明：

nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-8 运行环境



📖 说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.6 Atlas 200I SoC A1

Atlas 200I SoC A1（Atlas 200I SoC A1A200I SoC A1 核心板）开发和运行环境安装参考如下。

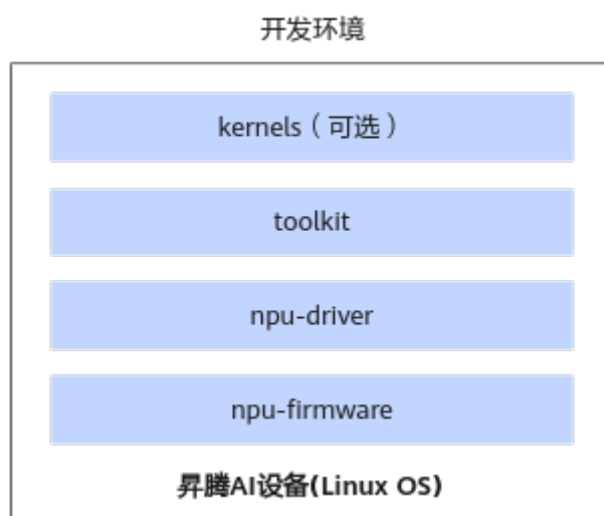
- 开发环境

开发环境安装软件如[图2-9](#)所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

图 2-9 开发环境



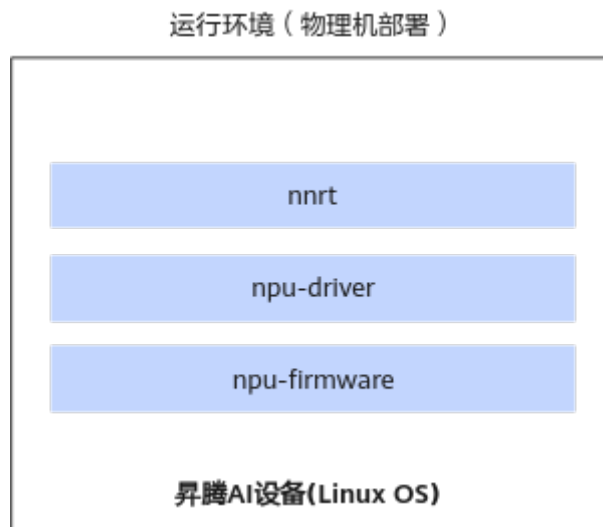
- 运行环境

运行环境安装软件如[图2-10](#)所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-10 运行环境



说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.7 A300-3000、A300-3010

A300-3000（Atlas 300I 推理卡（型号 3000））、A300-3010（Atlas 300I 推理卡（型号 3010））开发和运行环境安装参考如下：

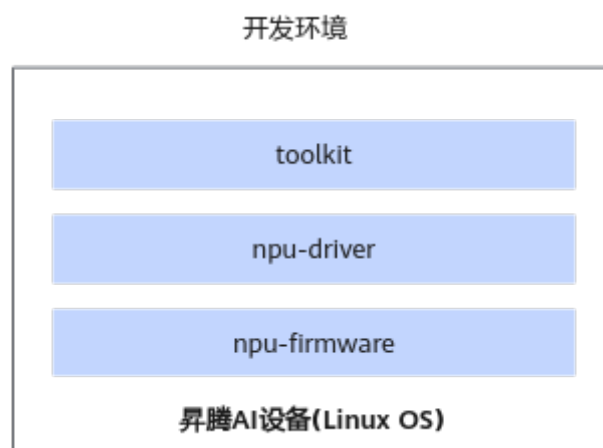
- 开发环境

开发环境安装软件如[图2-11](#)所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。

图 2-11 开发环境



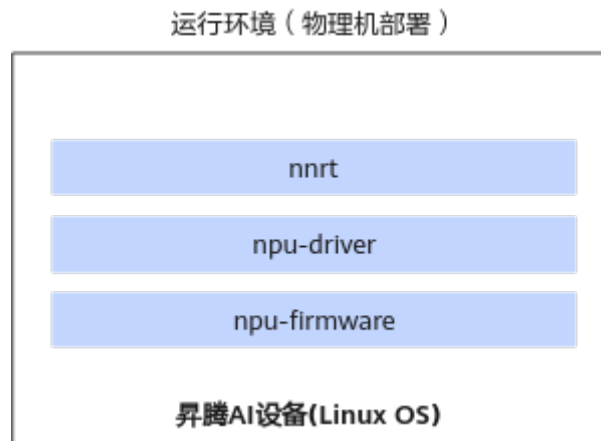
- 运行环境

运行环境安装软件如[图2-12](#)所示。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-12 运行环境



说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.8 Atlas 300I Pro、Atlas 300V Pro、Atlas 300V、Atlas 300I Duo

Atlas 300I Pro（Atlas 300I Pro 推理卡）、Atlas 300V Pro（Atlas 300V Pro 视频解析卡）、Atlas 300V（Atlas 300V 视频解析卡）、Atlas 300I Duo（Atlas 300I Duo 推理卡）开发和运行环境安装参考如下：

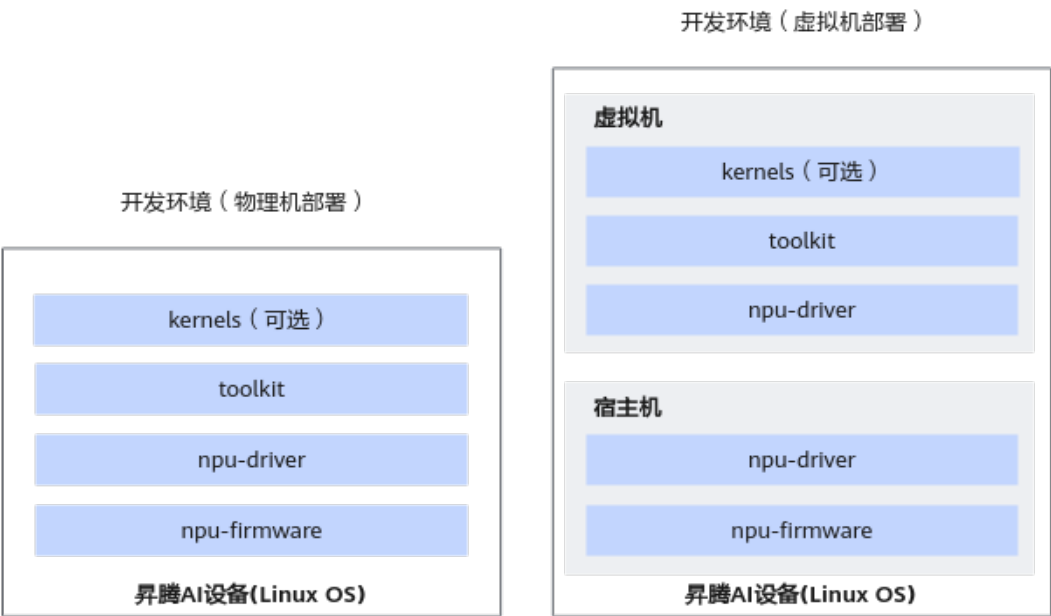
- 开发环境

开发环境安装软件如[图2-13](#)所示。

软件包说明：

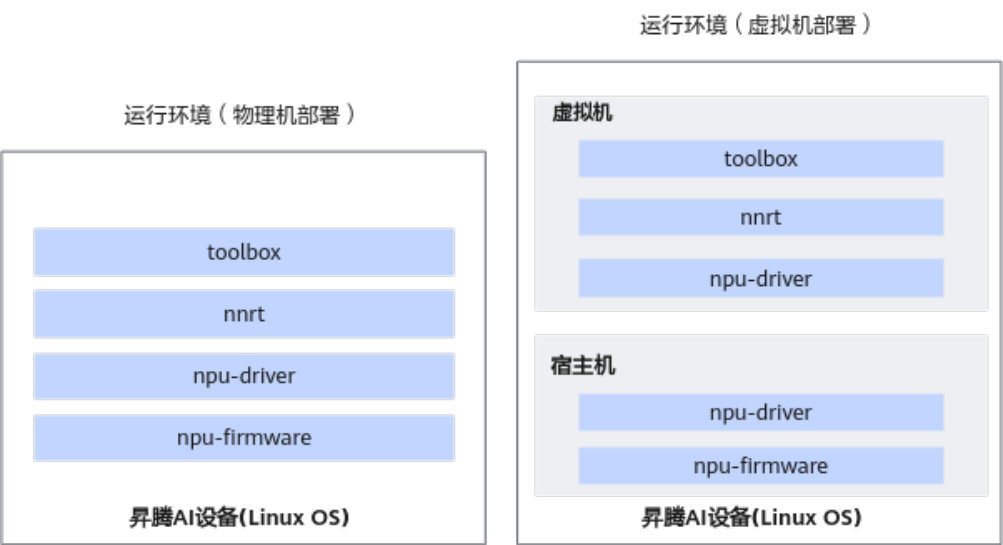
- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

图 2-13 开发环境



- 运行环境
运行环境安装软件如图2-14所示。
软件包说明：
 - npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-14 运行环境



说明

离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.9 Atlas 500 A2

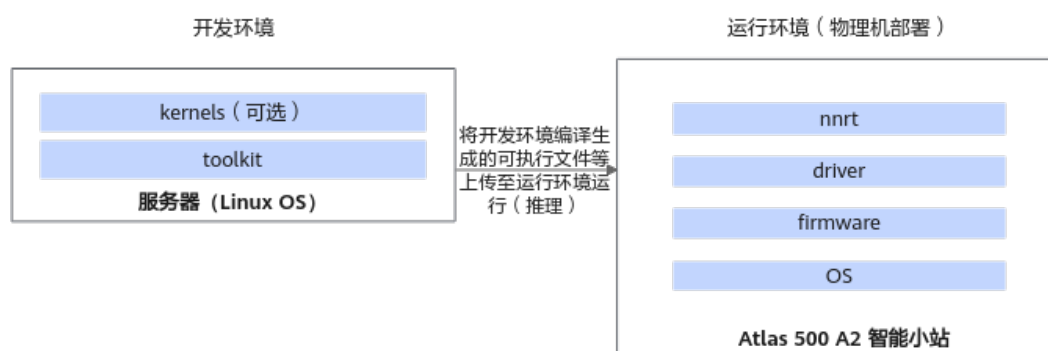
Atlas 500 A2（Atlas 500 A2 智能小站A500 A2 智能小站）仅支持作为运行环境。

- 开发环境
需要另外准备一台服务器用于搭建开发环境，详情参考[2.1 非昇腾设备](#)。
- 运行环境
 - 预置系统场景：Atlas 500 A2 智能小站A500 A2 智能小站出厂预安装Euler OS、驱动&固件，此场景下只支持通过容器运行推理应用。
 - 定制系统场景：Atlas 500 A2 智能小站支持挂载M.2外接存储，用户可以在外接存储中安装Ubuntu或openEuler操作系统，用户在制作镜像时合入驱动固件后，参考本文档在物理机上安装nnrt软件包。

Atlas 500 A2 智能小站A500 A2 智能小站初始配置请参考《[Atlas 500 A2 智能小站 用户指南](#)》，在容器内部署运行环境具体请参考《[Atlas 500 A2 智能小站 23.0.0 昇腾软件安装指南](#)》。

安装方案如[图2-15](#)所示。

图 2-15 Atlas 500 A2 智能小站

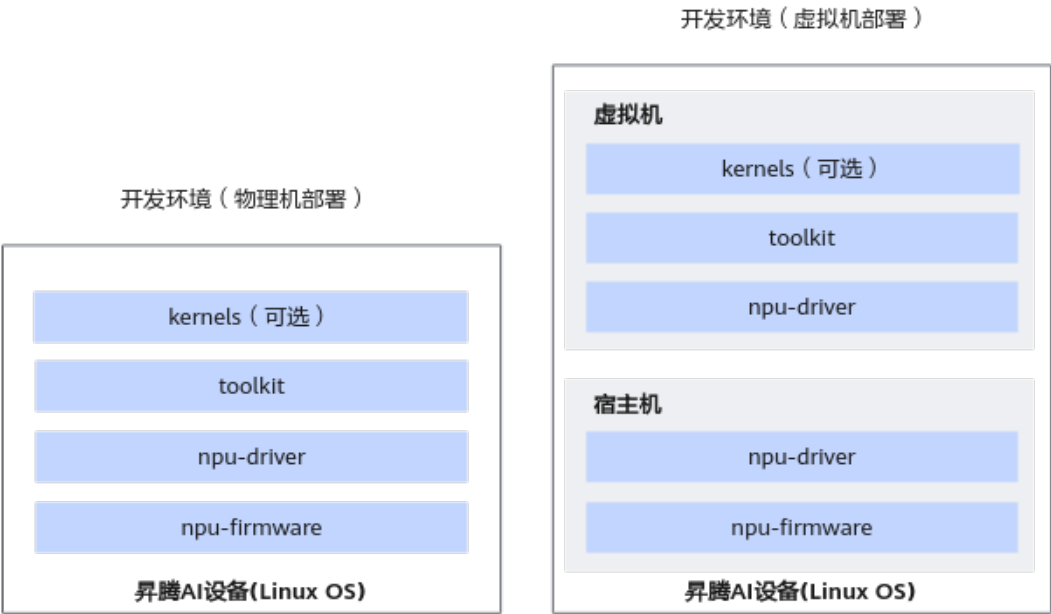


2.10 A500 Pro-3000

A500 Pro-3000（Atlas 500 Pro 智能边缘服务器（型号 3000））开发或运行环境安装参考如下：

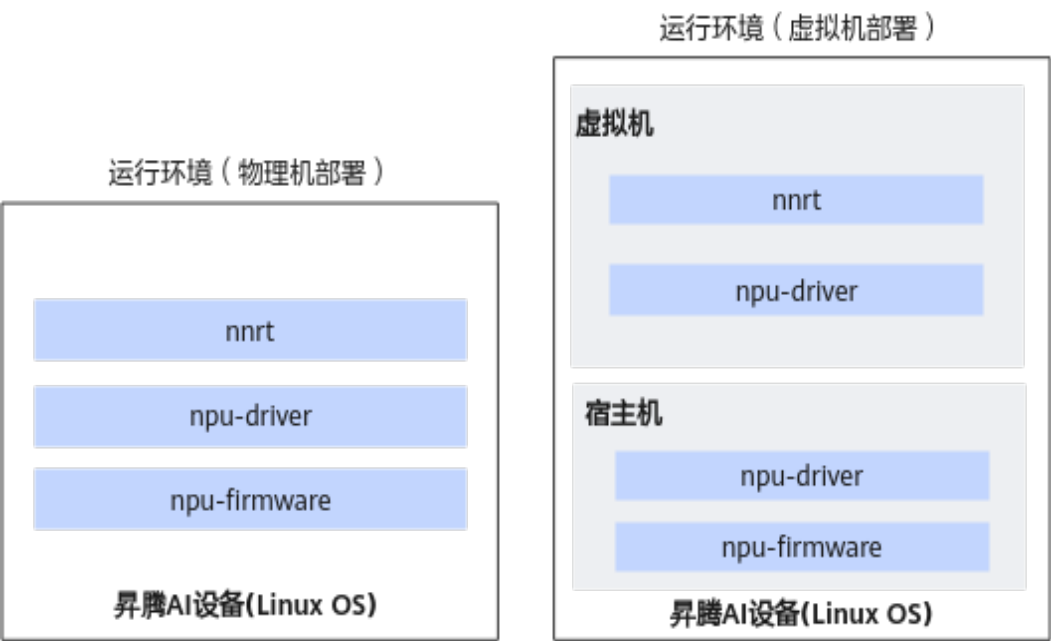
- 开发环境
开发环境安装软件如[图2-16](#)所示。
软件包说明：
 - npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
 - kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

图 2-16 开发环境



- 运行环境
运行环境安装软件如[图2-17](#)所示。
软件包说明：
 - npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - nnrt：离线推理引擎包，用于应用程序的模型推理。仅支持离线推理。

图 2-17 运行环境



说明

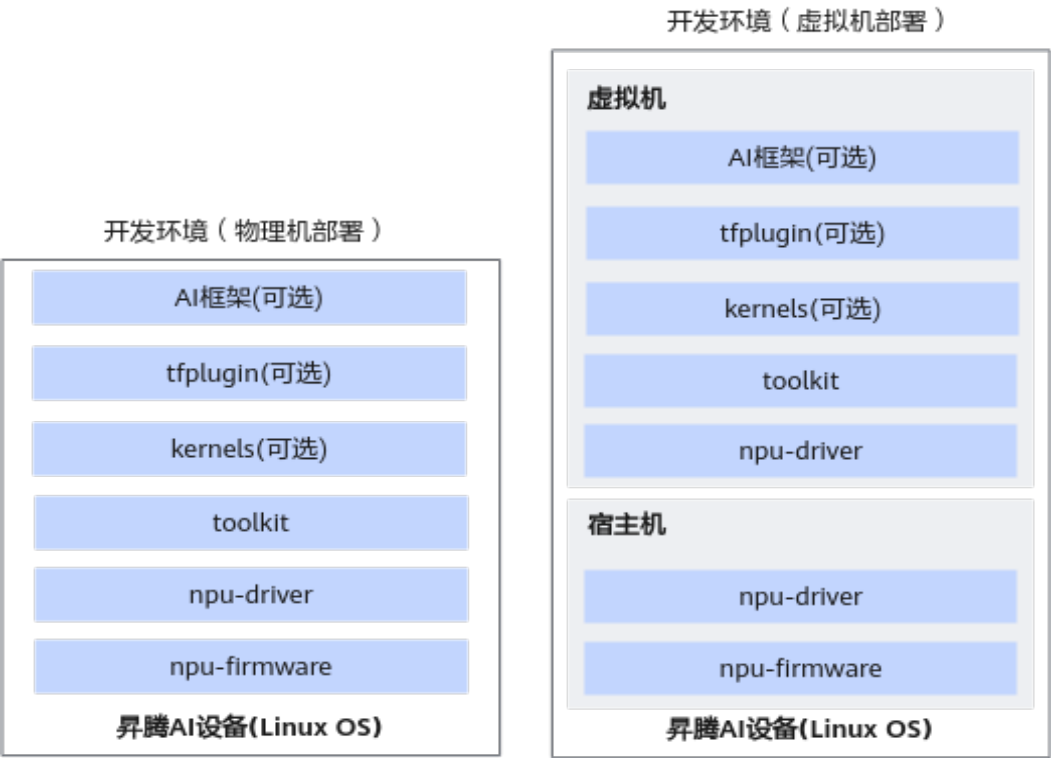
离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。

2.11 A800-3000、A800-3010

A800-3000（Atlas 800 推理服务器（型号 3000））、A800-3010（Atlas 800 推理服务器（型号 3010））开发或运行环境安装参考如下：

- 开发环境
开发环境安装软件如图2-18所示。
软件包说明：
 - npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
 - kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。
 - tfplugin（可选）：插件包，对接上层框架TensorFlow的适配插件。在线推理场景下若使用深度学习框架TensorFlow，需要安装该软件包。
 - AI框架（可选）：如需进行在线推理，则需安装AI框架。

图 2-18 开发环境

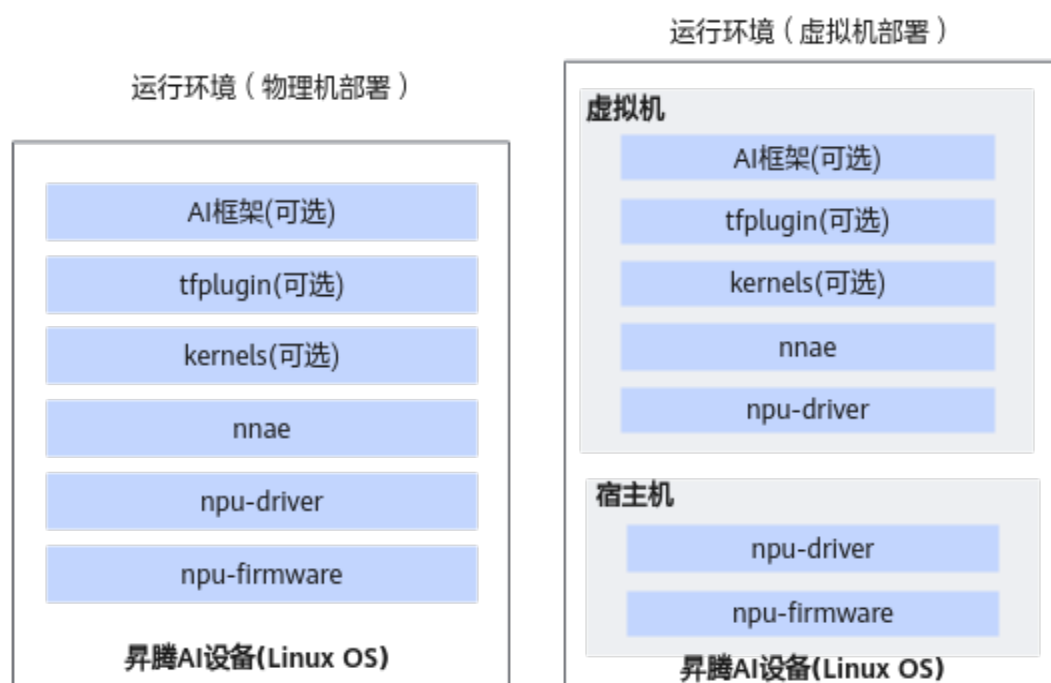


- 运行环境
运行环境安装软件如图2-19所示，其中nnae软件包支持离线推理。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnae：深度学习引擎包。支持离线推理。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。
- tfplugin（可选）：插件包，对接上层框架TensorFlow的适配插件。在线推理场景下若使用深度学习框架TensorFlow，需要安装该软件包。
- AI框架（可选）：如需进行在线推理，则需安装AI框架。

图 2-19 运行环境



说明

- 离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。
- 在线推理：是将原有AI框架做推理的应用快速迁移至昇腾AI处理器上，依赖于AI框架执行推理的场景。

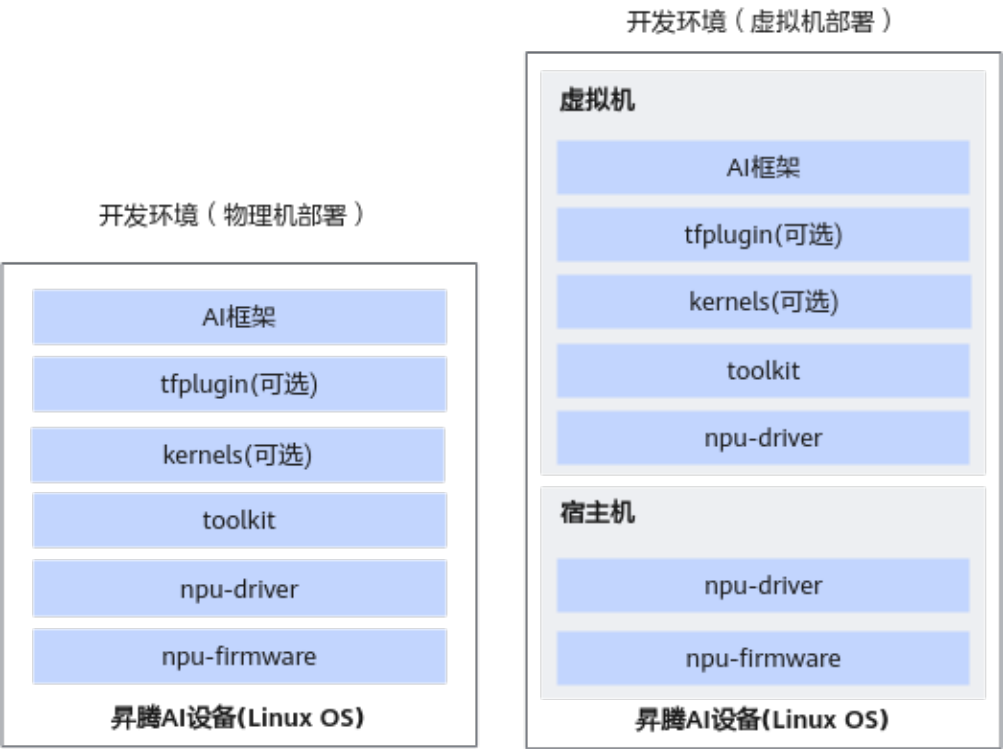
2.12 A300T-9000、Atlas 300T Pro、A800-9000、A800-9010、A900-9000

A300T-9000（Atlas 300T 训练卡（型号 9000））、Atlas 300T Pro（Atlas 300T Pro 训练卡（型号：9000））、A800-9000（Atlas 800 训练服务器（型号 9000））、A800-9010（Atlas 800 训练服务器（型号 9010））、A900-9000（Atlas 900 AI集群（型号 9000））开发或运行环境安装参考如下：

- 开发环境
开发环境安装软件如[图2-20](#)所示。

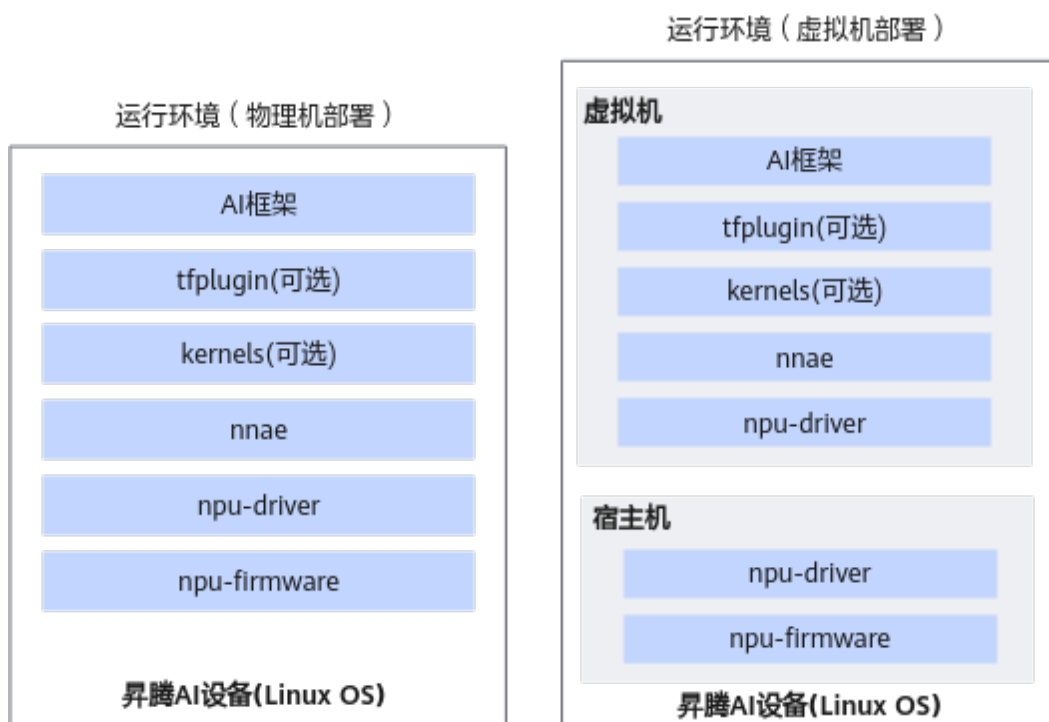
- 软件包说明：
- npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
 - kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。
 - tfplugin（可选）：插件包，对接上层框架TensorFlow的适配插件。在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装该软件包。
 - AI框架：深度学习框架。如MindSpore、TensorFlow、PyTorch等。

图 2-20 开发环境



- 运行环境
运行环境安装软件如[图2-21](#)所示。
软件包说明：
 - npu-firmware：固件安装包。
 - npu-driver：驱动安装包。
 - nnae：深度学习引擎包。支持离线推理、在线推理、训练。
 - kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。
 - tfplugin（可选）：插件包，对接上层框架TensorFlow的适配插件。在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装该软件包。
 - AI框架：深度学习框架。如MindSpore、TensorFlow、PyTorch等。

图 2-21 运行环境



说明

- 离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。
- 在线推理：是将原有AI框架做推理的应用快速迁移至昇腾AI处理器上，依赖于AI框架执行推理的场景。

2.13 Atlas 200T A2 Box16、Atlas 300T A2、Atlas 800T A2、Atlas 900 A2 PoD

Atlas 200T A2 Box16（Atlas 200T A2 Box16 异构子框）、Atlas 300T A2（Atlas 300T A2 训练卡A300T A2 训练卡）、Atlas 800T A2（Atlas 800T A2 训练服务器）、Atlas 900 A2 PoD（Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元）开发或运行环境安装参考如下：

• 开发环境

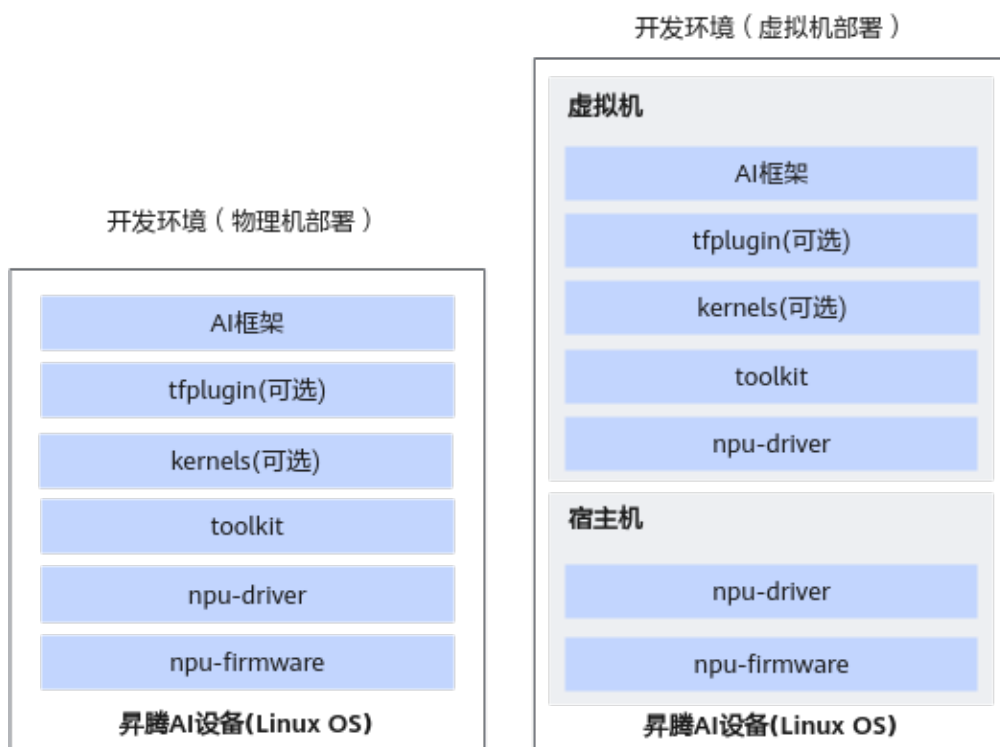
开发环境安装软件如图2-22所示，其中只有Atlas 800T A2 训练服务器、Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元支持虚拟机部署。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。
- kernels（可选）：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。

- tfplugin (可选)：插件包，对接上层框架TensorFlow的适配插件。在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装该软件包。
- AI框架：深度学习框架。如MindSpore、TensorFlow、PyTorch等。

图 2-22 开发环境



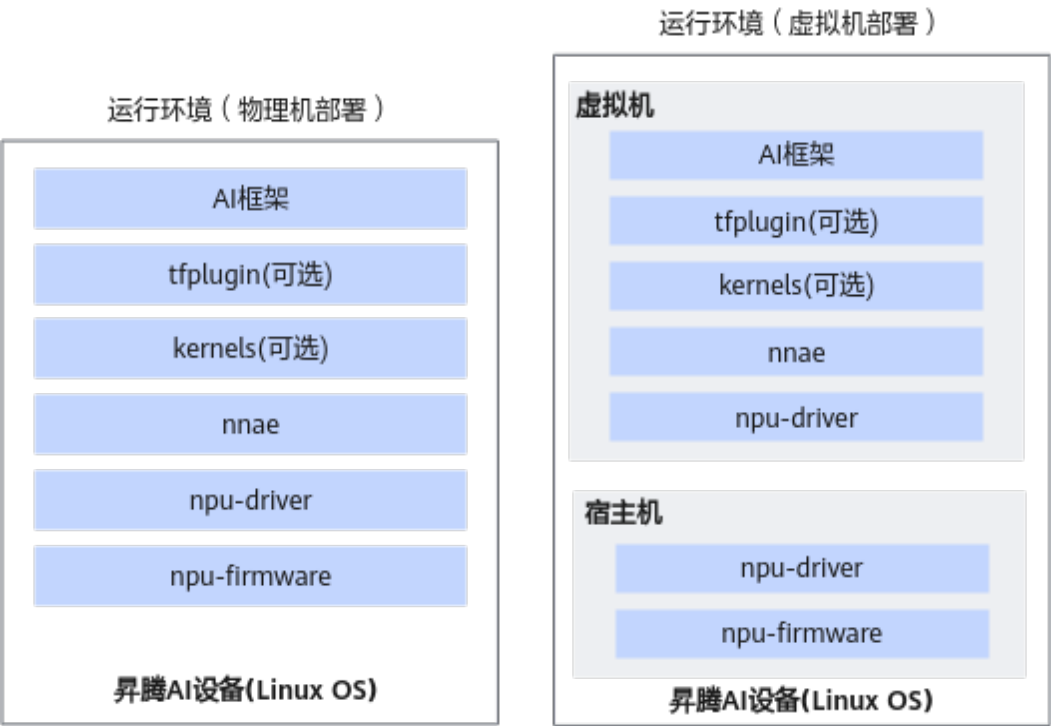
- 运行环境

运行环境安装软件如图2-23所示，其中只有Atlas 800T A2 训练服务器、Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元支持虚拟机部署。

软件包说明：

- npu-firmware：固件安装包。
- npu-driver：驱动安装包。
- nnae：深度学习引擎包。支持离线推理、在线推理、训练。
- kernels (可选)：二进制算子包。在包含动态shape网络或单算子API（例如aclnn类API）场景下使用。
- tfplugin (可选)：插件包，对接上层框架TensorFlow的适配插件。在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装该软件包。
- AI框架：深度学习框架。如MindSpore、TensorFlow、PyTorch等。

图 2-23 运行环境



说明

- 离线推理：是基于原有AI框架模型转换OM模型，不依赖于AI框架执行推理的场景。
- 在线推理：是将原有AI框架做推理的应用快速迁移至昇腾AI处理器上，依赖于AI框架执行推理的场景。

3 准备硬件环境

请先参考《昇腾产品形态说明》了解昇腾硬件产品，再根据表3-1，参考对应指导文档安装硬件产品和操作系统。

表 3-1 准备硬件环境

产品型号	参考手册
Atlas 200（EP场景）	- Atlas 200 AI加速模块安装的相关操作具体请参考《Atlas 200 AI加速模块 用户指南》。 - 操作系统要求请参考《Atlas 200 AI加速模块 6.0.0 NPU驱动和固件安装指南（EP场景）》。
Atlas 200（RC场景）	- Atlas 200 AI加速模块安装的相关操作具体请参考《Atlas 200 AI加速模块 用户指南》。 - 制作并安装启动镜像包具体请参考《Atlas 200 AI加速模块 6.0.0 软件安装与维护指南（RC场景）》。
Atlas 200I A2（EP场景）	- Atlas 200I A2 加速模块A200I A2 加速模块安装的相关操作具体请参考《Atlas 200I A2 加速模块 用户指南》。 - 操作系统要求请参考《NPU驱动和固件安装指南（EP场景）》。
Atlas 200I A2（RC场景）	- Atlas 200I A2 加速模块A200I A2 加速模块安装的相关操作具体请参考《Atlas 200I A2 加速模块 用户指南》。 - 制作并安装启动镜像包具体请参考《Atlas 200I A2 加速模块 23.0.0 昇腾软件安装指南》。
Atlas 500 A2	初始配置、制作并安装启动镜像包具体请参考《Atlas 500 A2 智能小站 用户指南》。

产品型号	参考手册
Atlas 200I SoC A1	- Atlas 200I SoC A1A200I SoC A1安装的相关操作具体请参考《Atlas 200I SoC A1 核心板 用户指南》。 - 操作系统要求请参考《Atlas 200I Soc A1 核心板 openEuler 20.03 LTS SP3 操作系统 安装指导书》。
A300-3000	- Atlas 300I 推理卡（型号：3000）的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300I 推理卡 用户指南 (型号 3000)》。
A300-3010	- Atlas 300I 推理卡（型号：3010）的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300I 推理卡 用户指南 (型号 3010)》。
Atlas 300I Pro	- Atlas 300I Pro 推理卡的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300I Pro 推理卡 用户指南》。
Atlas 300V Pro	- Atlas 300V Pro 视频解析卡的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300V Pro 视频解析卡 用户指南》。
Atlas 300V	- Atlas 300V 视频解析卡的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300V 视频解析卡 用户指南》。
Atlas 300I Duo	- Atlas 300I Duo 推理卡的安装方法可参见各服务器用户指南。 - 操作系统要求请参考《Atlas 300I Duo 推理卡 用户指南》。
Atlas 200T A2 Box16	- Atlas 200T A2 Box16 异构子框的安装方法可参见《Atlas 200T Box A2 异构子框 用户指南》。 - 操作系统要求请参考《Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件安装指南》。
Atlas 300T A2	- Atlas 300T A2 训练卡A300T A2 训练卡安装的相关操作具体请参考《Atlas 300T A2 训练卡 用户指南》。 - 操作系统要求请参考《Atlas 300T A2 训练卡 用户指南》。

产品型号	参考手册
A300T-9000	- Atlas 300T 训练卡（型号：9000）安装的相关操作具体请参考《Atlas 300T 训练卡 用户指南（型号 9000）》。 - 操作系统要求请参考《Atlas 300T 训练卡 用户指南（型号9000）》。
Atlas 300T Pro	- Atlas 300T Pro 训练卡（型号：9000）安装的相关操作具体请参考《Atlas 300T Pro 训练卡 用户指南（型号 9000）》。 - 操作系统要求请参考《Atlas 300T Pro 训练卡 用户指南（型号 9000）》。
A500 Pro-3000	- Atlas 500 Pro 智能边缘服务器（型号：3000）安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 500 Pro 智能边缘服务器 用户指南（型号 3000）》。 - 操作系统要求请参考《Atlas 500 Pro 智能边缘服务器 用户指南（型号 3000）》。
A800-3000	- Atlas 800 推理服务器（型号：3000）安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 800 推理服务器 用户指南（型号 3000）》。 - 操作系统要求请参考《Atlas 800 推理服务器 用户指南（型号 3000）》。
A800-3010	- Atlas 800 推理服务器（型号：3010）安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 800 推理服务器 用户指南（型号 3010）》。 - 操作系统要求请参考《Atlas 800 推理服务器 用户指南（型号 3010）》。
A800-9000	- Atlas 800 训练服务器（型号：9000）安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 800 训练服务器 用户指南（型号9000, 风冷）》或《Atlas 800 训练服务器 用户指南（型号 9000, 液冷）》。 - 操作系统要求请参考《Atlas 800 训练服务器 用户指南（型号9000, 风冷）》或《Atlas 800 训练服务器 用户指南（型号9000, 液冷）》。
Atlas 800T A2	- Atlas 800T A2 训练服务器安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 800T A2 训练服务器 用户指南》。 - 操作系统要求请参考《Atlas 800T A2 训练服务器 用户指南》。

产品型号	参考手册
A800-9010	- Atlas 800 训练服务器（型号：9010）安装上架、服务器基础参数配置、安装操作系统等操作请参见《Atlas 800 训练服务器 用户指南 (型号 9010)》。 - 操作系统要求请参考《Atlas 800 训练服务器 用户指南 (型号9010)》。
A900-9000	- Atlas 900 AI集群（型号：9000）安装上架、服务器基础参数配置、安装操作系统等操作，请根据集群配置参见对应的手册： 《Atlas 900 PoD 用户指南 (型号9000, 交流)》 《Atlas 900 计算节点 用户指南 (液冷)》 - 操作系统要求请参考《Atlas 900 PoD 用户指南 (型号9000, 交流)》或《Atlas 900 计算节点 用户指南 (液冷)》。
Atlas 900 A2 PoD	- Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元安装上架、服务器基础参数配置、安装操作系统等操作，请参见《Atlas 900 A2 PoD 集群基础单元 用户指南 (AICC场景)》 - 操作系统要求请参见《Atlas 900 A2 PoD 集群基础单元 用户指南 (AICC场景)》。

说明

操作系统镜像请从官网获取，示例如下：

- Ubuntu 22.04
 - aarch64
从Ubuntu官网<http://old-releases.ubuntu.com/releases/22.04/>下载ubuntu-22.04-live-server-arm64.iso。
 - x86_64
从Ubuntu官网<http://old-releases.ubuntu.com/releases/22.04/>下载ubuntu-22.04-live-server-amd64.iso。
- openEuler 22.03
 - aarch64
从openEuler官网<https://repo.openeuler.org/openEuler-22.03-LTS/ISO/aarch64/>下载openEuler-22.03-LTS-aarch64-dvd.iso
 - x86_64
从openEuler官网https://repo.openeuler.org/openEuler-22.03-LTS/ISO/x86_64/下载openEuler-22.03-LTS-x86_64-dvd.iso。

4 安装驱动和固件

- Atlas 200（RC场景）请参考《[Atlas 200 AI加速模块 6.0.0 软件安装与维护指南（RC场景）](#)》制作并安装启动镜像包。
- Atlas 200I A2 加速模块A200I A2 加速模块（RC场景）请参考《[Atlas 200I A2 加速模块 23.0.x\(23.0.3及其系列版本\) 驱动开发指南](#)》制作并安装好启动镜像包。
- Atlas 500 A2 智能小站A500 A2 智能小站请参考《[Atlas 500 A2 智能小站 用户指南](#)》制作并安装好启动镜像包。
- 其他昇腾AI处理器请先参考《[昇腾产品形态说明](#)》了解待安装设备所属的产品分类，再根据[表4-1](#)参考对应文档进行安装操作。

 说明

首次安装请按照“驱动->固件”的顺序，分别安装驱动和固件；覆盖安装请按照“固件->驱动”的顺序，分别安装固件和驱动。

表 4-1 驱动和固件安装指导

产品型号	参考文档
Atlas 200（EP场景）	请参见《 Atlas 200 AI加速模块 6.0.0 NPU驱动和固件安装指南（EP场景） 》。
Atlas 200I A2（EP场景）	请参见《 NPU驱动和固件安装指南（EP场景） 》。
Atlas 200I SoC A1	请参见《 Atlas 200I SoC A1核心板 23.0.3 NPU驱动和固件安装指南 》。
A300-3000	请参见《 Atlas 300I 推理卡 用户指南（型号 3000） 》的“安装与维护”章节。
A300-3010	请参见《 Atlas 300I 推理卡 用户指南（型号 3010） 》的“安装与维护”章节。
Atlas 300I Pro	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件安装指南 》。
Atlas 300V Pro	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件安装指南 》。

产品型号	参考文档
Atlas 300V	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件安装指南 》。
Atlas 300I Duo	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件安装指南 》。
Atlas 200T A2 Box16	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件安装指南 》。
Atlas 300T A2	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件安装指南 》。
A300T-9000	请参见《 Atlas 中心训练卡 23.0.3 NPU驱动和固件安装指南 》。
Atlas 300T Pro	请参见《 Atlas 中心训练卡 23.0.3 NPU驱动和固件安装指南 》。
A500 Pro-3000	请参见所配置标卡对应的文档。
A800-3000	请参见所配置标卡对应的文档。
A800-3010	请参见所配置标卡对应的文档。
A800-9000	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件安装指南 》。
A800-9010	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件安装指南 》。
Atlas 800T A2	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件安装指南 》。
A900-9000	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件安装指南 》。
Atlas 900 A2 PoD	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件安装指南 》。

5 CANN 软件包支持的操作系统

本章节提供软件包支持的操作系统清单，请执行以下命令查询当前操作系统，如果查询的操作系统版本不在对应产品列表中，请替换为支持的操作系统。

```
uname -m && cat /etc/*release
```

Atlas 200/300/500 推理产品

产品型号	支持的操作系统
Atlas 200	RC: Ubuntu 16.04.3
A800-3000+A300-3000	Ubuntu 20.04 EulerOS 2.8、EulerOS 2.9、EulerOS 2.10、EulerOS 2.11、EulerOS 2.12 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 7.8、CentOS 8.2 Kylin V10、Kylin V10 SP1、Neo Kylin 7.6 UOS20、UOS20 1020e SLES 12.5
A800-3000+A300-3010	Ubuntu 20.04 EulerOS 2.8、EulerOS 2.9、EulerOS 2.10、EulerOS 2.11 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 Kylin V10 SP1 UOS20 1020e

产品型号	支持的操作系统
A800-3010+A300-3010	Ubuntu 16.04.5、Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.5、EulerOS 2.9、EulerOS 2.10 openEuler 20.03、openEuler 22.03 CentOS 7.4、CentOS 7.6、CentOS 7.8、CentOS 8.2 SLES 12.4、SLES 12.5 BCLinux 7.6 Kylin V10、Kylin V10 SP1
A500 Pro-3000+A300-3000	Ubuntu 16.04.5 EulerOS 2.8、EulerOS 2.9 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 Linx 6.0.90、Linx 6.0.100 Kylin V10 SP1 UOS20 SP1

Atlas 200I/500 A2 推理产品

产品型号	支持的操作系统
Atlas 200I A2	RC: Ubuntu 22.04、openEuler 22.03 EP: Kylin V10 GFB-Release-2204-Build03
Atlas 500 A2	Ubuntu 22.04 EulerOS 2.11 openEuler 22.03

Atlas 推理系列产品（Ascend 310P AI 处理器）

产品型号	支持的操作系统
Atlas 200I SoC A1	openEuler 20.03

产品型号	支持的操作系统
A800-3000+Atlas 300I Pro A800-3000+Atlas 300V Pro	Ubuntu 20.04 CULinux3.0 EulerOS 2.9、EulerOS 2.10、EulerOS 2.11、EulerOS 2.12 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.5 Kylin V10 SP1、Kylin V10 SP2 UOS20 1050e
A800-3010+Atlas 300I Pro A800-3010+Atlas 300V Pro	Ubuntu 20.04 EulerOS 2.9、EulerOS 2.10 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.0 SLES 12.5 Kylin V10 SP1
A800-3000 + Atlas 300V A800-3010 + Atlas 300V	Ubuntu 20.04 CentOS 7.8 openEuler22.03
A500 Pro-3000+Atlas 300I Pro A500 Pro-3000+Atlas 300V Pro A500 Pro-3000+Atlas 300V	Ubuntu 20.04 EulerOS 2.10 openEuler 20.03、openEuler 22.03 CentOS 7.6 Linx 6.0.90、Linx 6.0.100 Kylin V10 SP1
Atlas 300I Duo	EulerOS 2.12 Ubuntu 20.04 CentOS 7.8 Kylin V10 SP1 UOS20 1020e

Atlas 训练系列产品

产品型号	支持的操作系统
A800-3000+A300T-9000 A800-3000+Atlas 300T Pro	Ubuntu 20.04 EulerOS 2.8 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 Kylin V10 SP1 UOS20 1020e
A800-3010+A300T-9000 A800-3010+Atlas 300T Pro	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 Debian 9.9、Debian 10.0 Kylin V10 SP1
A800-9000	Ubuntu 20.04 EulerOS 2.8、EulerOS 2.10、EulerOS 2.11、EulerOS 2.12 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 BCLinux 7.6、BCLinux 7.7 Kylin V10、Kylin V10 SP1、Kylin V10 SP2 UOS20 1020e
A800-9010	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 openEuler 20.03、openEuler 22.03 CentOS 7.6、CentOS 8.2 Debian 9.9、Debian 10.0 BCLinux 7.6 Kylin V10 SP1

Atlas A2 训练系列产品

产品型号	支持的操作系统
Atlas 200T A2 Box16	Ubuntu 20.04.1、Ubuntu 22.04、Ubuntu 22.04.1 (5.16.20-051620-generic) Debian 10.0
A800-3000 + Atlas 300T A2	Ubuntu 22.04 openEuler 22.03 Kylin V10 SP3

产品型号	支持的操作系统
Atlas 800T A2	Ubuntu 22.04 CTyunOS 22.06 CULinux 3.0 EulerOS 2.12 openEuler 22.03 CentOS 7.6 Kylin V10 SP2、Kylin V10 SP3 Kylin V10 GFB-Release-2204-Build03 BCLinux-for-Euler-21.10
Atlas 900 A2 PoD	CULinux 3.0 EulerOS 2.10、EulerOS 2.12 openEuler 22.03 Kylin V10 SP2、Kylin V10 SP3 BCLinux-for-Euler-21.10

6 安装开发环境

- 6.1 准备软件包
- 6.2 准备安装及运行用户
- 6.3 安装依赖
- 6.4 在非昇腾设备上安装
- 6.5 在昇腾设备上安装

6.1 准备软件包

下载软件包

软件安装前，请参考[表6-1](#)获取所需软件包和对应的数字签名文件，各软件包版本号需要保持一致。

下载本软件即表示您同意[华为企业业务最终用户许可协议（EULA）](#)的条款和条件。

表 6-1 CANN 软件包

名称	软件包	说明	获取链接
开发套件包	Ascend-cann-toolkit_{version}_linux-{arch}.run	• 主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的库文件、开发工具如ATC模型转换工具。 • 请根据CPU架构（x86_64、aarch64）获取对应的软件包。 • 对于运行环境为aarch64而开发环境为x86_64的场景，只需获取开发环境架构的开发套件包。	获取链接

名称	软件包	说明	获取链接
二进制算子包	Ascend-cann-kernels- {chip_type}_{version}_linux.run	（可选）二进制算子包，可以节省算子编译时间。 在包含动态shape网络或单算子API（例如aclnn类API）场景下，需要获取该软件包。	获取链接
框架插件包	Ascend-cann-tfplugin_{version}_linux-{arch}.run	（可选）插件包，对接上层框架TensorFlow的适配插件。 在线推理或训练场景下若使用深度学习框架TensorFlow，需要获取该软件包。	获取链接

说明

{version}表示软件版本号，{arch}表示CPU架构，{chip_type}表示芯片类型，可在昇腾AI处理器执行npu-smi info查看。

软件数字签名验证

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的数字签名文件用于完整性验证。

在软件包下载之后，请参考《OpenPGP签名验证指南》，对下载的软件包进行PGP数字签名校验。如果校验失败，请不要使用该软件包，先联系华为技术支持工程师解决。

使用软件包安装/升级之前，也需要按上述过程先验证软件包的数字签名，确保软件包未被篡改。

运营商客户请访问：<http://support.huawei.com/carrier/digitalSignatureAction>

企业客户请访问：<https://support.huawei.com/enterprise/zh/tool/pgp-verify-TL1000000054>

6.2 准备安装及运行用户

- 运行用户：实际运行推理业务或执行训练的用户。
- 安装用户：实际安装软件包的用户。
 - 若使用root用户安装，支持所有用户运行相关业务。
 - 若使用非root用户安装，则安装及运行用户必须相同。
 - 已有非root用户，则无需再次创建。
 - 若想使用新的非root用户，则需要先创建该用户，请参见如下方法创建。

📖 说明

- 如果安装驱动时未携带“--install-for-all”，并且CANN软件包运行用户为非root，则该CANN软件包运行用户所属的属组必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 运行用户不建议为root用户属组，权限控制可能存在安全风险，请谨慎使用。

创建非root用户操作方法如下，如下命令请以root用户执行。

1. 创建非root用户。

```
groupadd usergroup  
useradd -g usergroup -d /home/username -m username -s /bin/bash
```

2. 设置非root用户密码。

```
passwd username
```

📖 说明

- 创建完运行用户后，请勿关闭该用户的登录认证功能。
- 设置的口令需符合口令复杂度要求（请参见[C.4 口令复杂度要求](#)）。密码有效期为90天，您可以在/etc/login.defs文件中修改有效期的天数，或者通过chage命令来设置用户的有效期限，详情请参见[A.6 设置用户有效期](#)。

6.3 安装依赖

6.3.1 安装前必读

安装CANN软件前需安装相关依赖。

本章节以Ubuntu 22.04、openEuler 22.03和SLES 12.5为例，详述依赖安装操作。其他系统可参考这三种系统进行安装。

- Ubuntu、Debian、UOS20、UOS20 SP1、Linux系统可参考Ubuntu 22.04进行安装。
- EulerOS、openEuler、BCLinux、BCLinux-for-Euler、CentOS、CTyunOS、Kylin、UOS20 1020e、Ant OS、CULinux系统可参考openEuler 22.03进行安装。
- SLES系统可参考SLES 12.5进行安装。

6.3.2 依赖列表

📖 说明

针对用户自行安装的开源软件，如Python、numpy等，请使用稳定版本（尽量使用无漏洞的版本）。

Ubuntu 22.04

表 6-2 依赖信息

名称	版本要求
Python	CANN支持Python3.7.x (3.7.0~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.7) 、Python3.10.x (3.10.0~3.10.12) ，其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 - TensorFlow1.15支持Python3.7.x (3.7.0~3.7.11) 。 - Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x (3.7.5~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.2) 。
cmake	要求3.5.1版本及以上cmake，若cmake版本低于3.5.1，请参见 A.3 安装3.5.2版本cmake 解决。
make	-
gcc	要求7.3.0版本及以上gcc，若gcc版本低于7.3.0，可参考 A.4 安装7.3.0版本gcc 进行安装。
g++	
zlib1g zlib1g-dev libsqlite3-dev openssl libssl-dev libffi-dev libbz2-dev libxslt1-dev unzip pciutils net-tools libblas-dev gfortran libblas3 liblapack-dev liblapack3 libopenblas-dev	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.19.2
decorator	>=4.4.0
sympy	>=1.5.1

名称	版本要求
cffi	>=1.12.3
protobuf	>=3.13.0
attrs pyyaml pathlib2 scipy requests psutil absl-py	无版本要求，安装的版本以pip源为准。

openEuler 22.03

表 6-3 依赖信息

名称	版本限制
Python	CANN支持Python3.7.x (3.7.0~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.7) 、Python3.10.x (3.10.0~3.10.12) ，其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 - TensorFlow1.15支持Python3.7.x (3.7.0~3.7.11) 。 - Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x (3.7.5~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.2) 。
cmake	要求3.5.1版本及以上cmake，若cmake版本低于3.5.1，请参见 A.3 安装3.5.2版本cmake 解决。
make	-
gcc	要求7.3.0版本及以上gcc，若gcc版本低于7.3.0，可参考 A.4 安装7.3.0版本gcc 进行安装。
g++	

名称	版本限制
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools sqlite-devel lapack-devel gcc-gfortran python3-devel	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.19.2
decorator	>=4.4.0
sympy	>=1.4
cffi	>=1.12.3
protobuf	>=3.13.0
attrs pyyaml pathlib2 scipy requests psutil absl-py	无版本要求，安装的版本以pip源为准。

Suse 12SP5

表 6-4 依赖信息

类别	版本限制
Python	CANN支持Python3.7.x (3.7.0~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.7) 、Python3.10.x (3.10.0~3.10.12) ，其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 - TensorFlow1.15支持Python3.7.x (3.7.0~3.7.11) 。 - Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x (3.7.5~3.7.11) 、Python3.8.x (3.8.0~3.8.11) 、Python3.9.x (3.9.0~3.9.2) 。

类别	版本限制
cmake	>=3.5.1
make	-
gcc	离线推理场景： 要求4.8.5版本及以上gcc。
gcc-c++	
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools gdbm-devel	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.14.3
decorator	>=4.4.0
sympy	>=1.4
cffi	>=1.12.3
protobuf	>=3.11.3
attrs pyyaml pathlib2 scipy requests psutil gnureadline absl-py	无版本要求，安装的版本以pip源为准。

6.3.3 安装步骤（Ubuntu 22.04）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
apt-get update
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，则检查网络是否连接或者把“/etc/apt/sources.list”文件中的源更换为可用的源或使用镜像源（以配置华为镜像源为例，可参考[华为开源镜像站](#)）。

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```
3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。
 - a. 在任意目录下执行如下命令，打开**.bashrc**文件：

```
vi ~/.bashrc
```

在文件最后一行后面添加umask 0022内容。
 - b. 执行**wq!**命令保存文件并退出。
 - c. 执行**source ~/.bashrc**命令使其立即生效。

说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中**umask 0022**一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
gcc --version
g++ --version
make --version
cmake --version
dpkg -l zlib1g| grep zlib1g| grep ii
dpkg -l zlib1g-dev| grep zlib1g-dev| grep ii
dpkg -l libsqlite3-dev| grep libsqlite3-dev| grep ii
dpkg -l openssl| grep openssl| grep ii
dpkg -l libssl-dev| grep libssl-dev| grep ii
dpkg -l libffi-dev| grep libffi-dev| grep ii
dpkg -l libbz2-dev| grep libbz2-dev| grep ii
dpkg -l libxslt1-dev| grep libxslt1-dev| grep ii
dpkg -l unzip| grep unzip| grep ii
dpkg -l pciutils| grep pciutils| grep ii
dpkg -l net-tools| grep net-tools| grep ii
dpkg -l libblas-dev| grep libblas-dev| grep ii
dpkg -l gfortran| grep gfortran| grep ii
dpkg -l libblas3| grep libblas3| grep ii
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[6.3.2 依赖列表](#)为准）。

```
gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
g++ (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
GNU Make 4.3
cmake version 3.22.1
ii  zlib1g:arm64  1:1.2.11.dfsg-2ubuntu9 arm64      compression library - runtime
ii  zlib1g-dev:arm64 1:1.2.11.dfsg-2ubuntu9 arm64      compression library - development
ii  libsqlite3-dev:arm64 3.37.2-2 arm64      SQLite 3 development files
ii  openssl      3.0.2-0ubuntu1 arm64      Secure Sockets Layer toolkit - cryptographic utility
ii  libssl-dev:arm64 3.0.2-0ubuntu1 arm64      Secure Sockets Layer toolkit - development files
```

```

ii libffi-dev:arm64 3.4.2-4 arm64 Foreign Function Interface library (development files)
ii libbz2-dev:amd64 1.0.6-8.1ubuntu0.2 arm64 high-quality block-sorting file compressor library -
development
ii libxslt1-dev:amd64 1.1.29-5ubuntu0.3 arm64 XSLT 1.0 processing library - development kit
ii unzip 6.0-26ubuntu3 arm64 De-archiver for .zip files
ii pciutils 1:3.7.0-6 arm64 PCI utilities
ii net-tools 1.60+git201811103.0eebece-1ubuntu5 arm64 NET-3 networking toolkit
libblas-dev:arm64 3.9.0-1build1 arm64 Basic Linear Algebra Subroutines 3, static library
gfortran 4:9.3.0-1ubuntu2 arm64 GNU Fortran 95 compiler
libblas3:arm64 3.9.0-1build1 arm64 Basic Linear Algebra Reference implementations, shared library

```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将**步骤1至步骤2**命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行**su - username**命令切换到非root用户继续执行**步骤1至步骤3**。
- libsqlite3-dev需要在python安装之前安装，如果用户操作系统已经安装满足版本要求的python环境，在此之后再安装libsqlite3-dev，则需要重新编译python环境。

```

sudo apt-get install -y gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev
libbz2-dev libxslt1-dev unzip pciutils net-tools libblas-dev gfortran libblas3

```

步骤2 检查系统是否安装满足版本要求的python开发环境，具体要求请参见[6.3.2 依赖列表](#)。

执行如下命令，如果返回信息满足python版本要求，则直接进入下一步，否则可参考[A.2 编译安装Python3.7.5](#)。

```

python3 --version
pip3 --version

```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```

pip3 install attrs
pip3 install numpy
pip3 install decorator
pip3 install sympy
pip3 install cffi
pip3 install pyyaml
pip3 install pathlib2
pip3 install psutil
pip3 install protobuf
pip3 install scipy
pip3 install requests
pip3 install absl-py

```

如果执行上述命令时报错“subprocess.CalledProcessError: Command '('lsb_release', '-a')' return non-zero exit status 1”，请参见[B.6 pip3 install报错“subprocess.CalledProcessError: Command '\('lsb_release', '-a'\)' return non-zero exit status 1”](#)。

----结束

📖 说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

6.3.4 安装步骤（openEuler 22.03）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
yum makecache
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，则检查网络是否连接或者把"/etc/yum.repos.d/xxxx.repo"文件中的源更换为可用的源或使用镜像源（以配置华为镜像源为例，可参考[华为开源镜像站](#)）。

📖 说明

如果执行上述命令提示 “Your license is invalid”，请获取OS授权license。

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```
3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。
 - a. 在任意目录下执行如下命令，打开.bashrc文件：

```
vi ~/.bashrc
```

在文件最后一行后面添加umask 0022内容。
 - b. 执行:wq!命令保存文件并退出。
 - c. 执行source ~/.bashrc命令使其立即生效。

📖 说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

配置最大线程数

训练场景下，不同OS的最大线程数可能不满足训练要求，需执行以下命令修改最大线程数为无限制。

1. 以root用户登录安装环境。

2. 配置环境变量，修改线程数为无限制，编辑“/etc/profile”文件，在文件的最后添加如下内容后保存退出：
`ulimit -u unlimited`
3. 执行如下命令使环境变量生效。
`source /etc/profile`

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
gcc --version
g++ --version
make --version
cmake --version
rpm -qa |grep unzip
rpm -qa |grep zlib-devel
rpm -qa |grep libffi-devel
rpm -qa |grep openssl-devel
rpm -qa |grep pciutils
rpm -qa |grep net-tools
rpm -qa |grep sqlite-devel
rpm -qa |grep lapack-devel
rpm -qa |grep gcc-gfortran
rpm -qa |grep python3-devel
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[6.3.2 依赖列表](#)为准）。

```
gcc (GCC) 10.3.1
g++ (GCC) 10.3.1
GNU Make 4.3
cmake version 3.22.0
unzip-6.0-45.oe1.aarch64
zlib-devel-1.2.11-22.oe1.aarch64
libffi-devel-3.4.2-2.oe2203.aarch64
openssl-devel-1.1.1m-18.oe2203.aarch64
pciutils-3.7.0-2.oe2203.aarch64
net-tools-2.10-3.oe2203.aarch64
sqlite-devel-3.24.0-9.oe1.aarch64
lapack-devel-3.8.0-16.oe1.aarch64
gcc-gfortran-7.3.0-20190804.h31.oe1.aarch64
python3-devel-3.7.4-8.oe1.aarch64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将**步骤1至步骤2**命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行**su - username**命令切换到非root用户继续执行**步骤1至步骤3**。
- sqlite-devel需要在python安装之前安装，如果用户操作系统已经安装满足版本要求的python环境，在此之后再安装sqlite-devel，则需要重新编译python环境。

```
sudo yum install -y gcc gcc-c++ make cmake unzip zlib-devel libffi-devel openssl-devel pciutils net-tools
sqlite-devel lapack-devel gcc-gfortran python3-devel
```

如果通过上述方式安装的cmake版本低于3.5.1，则请参见[A.3 安装3.5.2版本cmake](#)解决。

步骤2 检查系统是否安装满足版本要求的python开发环境，具体要求请参见[6.3.2 依赖列表](#)。

执行如下命令，如果返回信息满足python版本要求，则直接进入下一步，否则可参考[A.2 编译安装Python3.7.5](#)。

```
python3 --version
pip3 --version
```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```
pip3 install attrs
pip3 install numpy
pip3 install decorator
pip3 install sympy
pip3 install cffi
pip3 install pyyaml
pip3 install pathlib2
pip3 install psutil
pip3 install protobuf
pip3 install scipy
pip3 install requests
pip3 install absl-py
```

- 如果安装numpy报错，请参考[B.5 pip3 install numpy报错解决](#)。
- 如果安装scipy报错，请参考[B.4 pip3 install scipy报错解决](#)。

步骤4 如果仅需支持离线推理，请跳过此步骤。

CentOS7.6系统使用软件源默认安装的gcc版本为4.8.5，因此需要安装7.3.0版本gcc，安装过程请参见[A.4 安装7.3.0版本gcc](#)。

----结束

说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

6.3.5 安装步骤（SLES 12.5）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
zypper ref
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，可参考以下操作配置本地源。

请以root用户执行如下操作。

1. 挂载系统镜像iso文件（以下命令为示例）。

```
mount -o loop SLE-12-SP5-Server-DVD-x86_64-GM-DVD1.iso /mnt
```
2. 使用ar参数添加一个mnt路径下挂好的本地源，并命名为suse（用户可自定义）。

```
zypper ar -f /mnt suse
```

3. 清空zypper缓存，刷新源，使配置的源生效。

```
zypper clean
zypper ref
```

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```

3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。

- a. 在任意目录下执行如下命令，打开.bashrc文件：

```
vi ~/.bashrc
```

在文件最后一行后面添加umask 0022内容。

- b. 执行:wq!命令保存文件并退出。
- c. 执行source ~/.bashrc命令使其立即生效。

说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
rpm -qa | grep gcc
rpm -qa | grep make
rpm -qa | grep unzip
rpm -qa | grep zlib-devel
rpm -qa | grep openssl-devel
rpm -qa | grep pciutils
rpm -qa | grep net-tools
rpm -qa | grep gdbm-devel
rpm -qa | grep libffi-devel
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[6.3.2 依赖列表](#)为准）。

```
gcc48-4.8.5-31.20.1.x86_64
libgcc_s1-8.2.1+r264010-1.3.3.x86_64
gcc-4.8-6.189.x86_64
libgcc_s1-32bit-8.2.1+r264010-1.3.3.x86_64
gcc-c++-4.8-6.189.x86_64
gcc48-c++-4.8.5-31.20.1.x86_64
cmake-3.5.2-20.6.1.x86_64
makedumpfile-1.6.5-1.19.x86_64
automake-1.13.4-6.2.noarch
make-4.0-4.1.x86_64
unzip-6.00-33.8.1.x86_64
zlib-devel-1.2.11-3.21.1.x86_64
```

```
zlib-devel-32bit-1.2.11-3.21.1.x86_64
zlib-devel-static-1.2.11-3.21.1.x86_64
zlib-devel-1.2.11-9.42.x86_64
zlib-devel-static-32bit-1.2.11-3.21.1.x86_64
libopenssl-devel-1.0.2p-1.13.noarch
pciutils-3.2.1-11.3.1.x86_64
pciutils-ids-2018.02.08-12.3.1.noarch
net-tools-1.60-765.5.4.x86_64
gdbm-devel-1.10-9.70.x86_64
libffi-devel-3.2.1.git259-10.8.x86_64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将**步骤1至步骤2**命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行**su - username**命令切换到非root用户继续执行**步骤1至步骤3**。

```
sudo zypper install -y gcc gcc-c++ make cmake unzip zlib-devel openssl-devel pciutils net-tools gdbm-devel
```

由于本地源中缺少libffi-devel依赖，可从[opensesame镜像源](#)中下载libffi-devel-3.2.1.git259-10.8.x86_64.rpm、libffi7-3.2.1.git259-10.8.x86_64.rpm、libffi7-32bit-3.2.1.git259-10.8.x86_64.rpm（软件包会定时更新，请以实际rpm包名为准）并一同上传至服务器某一目录下（如“/home/test”）。

进入rpm包所在路径（如“/home/test”），执行如下命令安装所需依赖：

```
sudo rpm -ivh *.rpm --nodeps
```

步骤2 检查系统是否安装满足版本要求的python开发环境（具体要求请参见[6.3.2 依赖列表](#)，此步骤以环境上需要使用python 3.7.x为例进行说明）。

执行命令**python3 --version**，如果返回信息满足python版本要求，则直接进入下一步。

否则可参考如下方式安装python3.7.5。

1. 使用wget下载python3.7.5源码包，可以下载到安装环境的任意目录，命令为：

```
wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
```

2. 进入下载后的目录，解压源码包，命令为：

```
tar -zxvf Python-3.7.5.tgz
```

3. 进入解压后的文件夹，创建安装目录，执行配置、编译和安装命令：

```
cd Python-3.7.5
./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
make
sudo make install
```

其中“--prefix”参数用于指定python安装路径，用户根据实际情况进行修改，“--enable-shared”参数用于编译出libpython3.7m.so.1.0动态库。

本手册以--prefix=/usr/local/python3.7.5路径为例进行说明。执行配置、编译和安装命令后，安装包在/usr/local/python3.7.5路径，libpython3.7m.so.1.0动态库在/usr/local/python3.7.5/lib/libpython3.7m.so.1.0路径。

4. 设置python3.7.5环境变量。

```
#用于设置python3.7.5库文件路径
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
#如果用户环境存在多个python3版本，则指定使用python3.7.5版本
export PATH=/usr/local/python3.7.5/bin:$PATH
```

通过以上export方式设置环境变量，该种方式设置的环境变量只在当前窗口有效。您也可以通过将以上命令写入~/.bashrc文件中，然后执行**source ~/.bashrc**

命令，使上述环境变量永久生效。注意如果后续您有使用环境上其他python版本的需求，则不建议将以上命令写入到~/.bashrc文件中。

5. 安装完成之后，执行如下命令查看安装版本，如果返回相关版本信息，则说明安装成功。

```
python3 --version
pip3 --version
```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```
pip3 install attrs
pip3 install numpy
pip3 install decorator
pip3 install sympy
pip3 install cffi
pip3 install pyyaml
pip3 install pathlib2
pip3 install psutil
pip3 install protobuf
pip3 install scipy
pip3 install requests
pip3 install gnureadline
pip3 install absl-py
```

----结束

说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

6.4 在非昇腾设备上安装

6.4.1 安装开发套件包

前提条件

- 请参见[6.3 安装依赖](#)完成安装前准备。
- 通过[6.1 准备软件包](#)章节获取开发环境架构的开发套件包Ascend-cann-toolkit_xxx.run。
- 安装开发套件包前请确保安装目录可用空间大于9G，如不满足请清理空间或更换安装目录。

安装步骤

步骤1 以软件包的安装用户登录安装环境。

若**6.3 安装依赖**中安装依赖的用户为root用户，则软件包的安装用户可自行指定；若**6.3 安装依赖**中安装依赖的用户为非root用户，请确保软件包的安装用户与该用户保持一致。

步骤2 将获取到的开发套件包上传到安装环境任意路径（如“/home/package”）。

步骤3 进入软件包所在路径。

步骤4 增加对软件包的可执行权限。

```
chmod +x 软件包名.run
```

其中**软件包名.run**表示开发套件包Ascend-cann-toolkit_{version}_linux-{arch}.run，请根据实际包名进行替换。

步骤5 执行如下命令校验软件包安装文件的一致性和完整性。

```
./软件包名.run --check
```

步骤6 执行如下命令安装软件（以下命令支持--install-path=<path>等参数，具体参数说明请参见**C.1 参数说明**）。

```
./软件包名.run --install
```

说明

- 用户想使用默认签署**华为企业业务最终用户许可协议（EULA）**的方式安装软件包时，可以添加--quiet参数配合安装命令使用，如：./软件包名.run --install --quiet 添加该参数后会跳过**步骤7**的确认操作。
- 如果以root用户安装，**不允许安装在非root用户目录下**。
- 如果用户未指定安装路径，则软件会安装到默认路径下，默认安装路径如下。
 - root用户：“/usr/local/Ascend”
 - 非root用户：“\${HOME}/Ascend”其中\${HOME}为当前用户目录。软件包支持多版本安装，同一安装路径下，可以安装多个版本的软件包，不同版本的软件通过版本目录来区分。默认使用最后安装的版本（即通过**软件包名/latest**建立软链接到最后安装的版本）。
- 软件包安装详细日志路径如下。
 - root用户：“/var/log/ascend_seclog/ascend_toolkit_install.log”
 - 非root用户：“\${HOME}/var/log/ascend_seclog/ascend_toolkit_install.log”其中\${HOME}为当前用户目录。

步骤7 用户需签署**华为企业业务最终用户许可协议（EULA）**后进入安装流程，根据回显页面执行**y**或**Y**确认协议，输入其他任意字符为拒绝协议，确认接受协议后开始安装。

若当前语言环境不满足要求，可以执行如下命令配置系统的默认语言环境。

```
#配置为中文（简体）
export LANG=zh_CN.UTF-8
#配置为英文
export LANG=en_US.UTF-8
```

安装完成后，若显示如下信息，则说明软件安装成功：

```
xxx install success
```

xxx表示安装的实际软件包名。

----结束

6.4.2 安装二进制算子包

在包含动态shape网络或单算子API（例如aclnn类API）场景下需要安装算子包Ascend-cann-kernels-xxx_linux.run，支持多个芯片类型的二进制算子包安装在同一台设备上，请通过**6.1 准备软件包**章节获取。

二进制算子包依赖CANN开发套件包Ascend-cann-toolkit_XXX.run，安装时确保已安装配套版本的toolkit，并使用同一用户安装。请参照[6.4.1 安装开发套件包](#)的步骤安装二进制算子包。

安装完成后所在路径如下（以默认安装路径为例），用户可以在该路径下查询是否已安装二进制算子包：

- root用户：“/usr/local/Ascend/ascend-toolkit/latest/opp/built-in/op_impl/ai_core/tbe/kernel”
- 非root用户：“\${HOME}/Ascend/ascend-toolkit/latest/opp/built-in/op_impl/ai_core/tbe/kernel”

其中\${HOME}为当前用户目录。

6.4.3 配置环境变量

CANN软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

```
# 安装toolkit包时配置
source /usr/local/Ascend/ascend-toolkit/set_env.sh
# 其中<arch>请替换为实际架构
export LD_LIBRARY_PATH=/usr/local/Ascend/ascend-toolkit/latest/<arch>linux/devlib:$LD_LIBRARY_PATH
```

用户也可以通过修改~/.bashrc文件方式设置永久环境变量，操作如下：

1. 以运行用户在任意目录下执行vi ~/.bashrc命令，打开.bashrc文件，在文件最后一行后面添加上述内容。
2. 执行:wq!命令保存文件并退出。
3. 执行source ~/.bashrc命令使其立即生效。

6.4.4 配置交叉编译环境

若开发环境和运行环境上的操作系统架构不同，需在开发环境中使用交叉编译工具，并使用运行环境架构的库文件进行编译，这样编译出来的可执行文件，才可以在运行环境中执行。

针对运行环境为Atlas 200I A2 加速模块A200I A2 加速模块（RC场景）和Atlas 500 A2 智能小站A500 A2 智能小站，若使用一台X86服务器（或者PC）搭建开发环境，则需要在开发环境使用交叉编译工具和aarch64架构的库文件进行编译。

当前支持使用开发套件包中提供的交叉编译工具和Linux上支持的g++交叉编译工具进行编译，运行环境为Atlas 200I A2 加速模块A200I A2 加速模块（RC场景）和Atlas 500 A2 智能小站A500 A2 智能小站时，CANN已适配的操作系统（如openEuler，Ubuntu等）可以使用开发套件包中提供的交叉编译工具，未适配的操作系统和其他昇腾AI处理器（如运行环境为Atlas 200 AI加速模块（RC场景））请用户自行安装g++交叉编译工具。具体如[表6-5](#)所示。

表 6-5 安装交叉编译工具

开发环境架构	运行环境架构	编译环境配置
x86_64	aarch64	- g++交叉编译工具：请使用软件包的安装用户，在开发环境执行aarch64-linux-gnu-g++ --version命令检查是否安装g++交叉编译工具，若已经安装则可以忽略。 安装命令示例如下（以下命令仅为示例，请用户根据实际情况替换，如果使用root用户安装依赖，请删除命令中的sudo）： <code>sudo apt-get install g++-aarch64-linux-gnu</code> - 开发套件包中交叉编译工具所在路径（以root用户默认安装路径为例）：<code>/usr/local/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin</code>。 - aarch64架构的库文件所在路径（以root用户默认安装路径为例）：<code>/usr/local/Ascend/ascend-toolkit/latest/runtime/lib64/stub/aarch64</code>。

6.5 在昇腾设备上安装

6.5.1 安装开发套件包

前提条件

- 请参见6.3 安装依赖完成安装前准备。
- 通过6.1 准备软件包章节获取开发套件包Ascend-cann-toolkit_xxx.run。
- 安装开发套件包前请确保安装目录可用空间大于9G，如不满足请清理空间或更换安装目录。

安装步骤

- 步骤1 以软件包的安装用户登录安装环境。

若6.3 安装依赖中安装依赖的用户为root用户，则软件包的安装用户可自行指定；若6.3 安装依赖中安装依赖的用户为非root用户，请确保软件包的安装用户与该用户保持一致。
- 步骤2 将获取到的开发套件包上传到安装环境任意路径（如“/home/package”）。
- 步骤3 进入软件包所在路径。
- 步骤4 增加对软件包的可执行权限。
`chmod +x 软件包名.run`

软件包名.run表示开发套件包Ascend-cann-toolkit_{version}_linux-{arch}.run，请根据实际情况包名进行替换。
- 步骤5 执行如下命令校验软件包安装文件的一致性和完整性。
`./软件包名.run --check`

步骤6 执行以下命令安装软件（以下命令支持`--install-path=<path>`等参数，具体参数说明请参见[C.1 参数说明](#)）。

```
./软件包名.run --install
```

📖 说明

- 开发套件包支持不同用户在同一开发环境安装，但安装版本必须保持一致，不同用户所属的属组必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 用户想使用默认签署[华为企业业务最终用户许可协议（EULA）](#)的方式安装软件包时，可以添加`--quiet`参数配合安装命令使用，如：`./软件包名.run --install --quiet` 添加该参数后会跳过[步骤7](#)的确认操作。
- 如果以root用户安装，**不允许安装在非root用户目录下**。
- 如果用户未指定安装路径，则软件会安装到默认路径下，默认安装路径如下。
 - root用户：“`/usr/local/Ascend`”
 - 非root用户：“`${HOME}/Ascend`”其中`${HOME}`为当前用户目录。软件包支持多版本安装，同一安装路径下，可以安装多个版本的软件包，不同版本的软件通过版本目录来区分。默认使用最后安装的版本（即通过 `软件包名/latest` 建立软连接到最后安装的版本）。
- 软件包安装详细日志路径如下。
 - root用户：“`/var/log/ascend_seclog/ascend_toolkit_install.log`”
 - 非root用户：“`${HOME}/var/log/ascend_seclog/ascend_toolkit_install.log`”其中`${HOME}`为当前用户目录。

步骤7 用户需签署[华为企业业务最终用户许可协议（EULA）](#)后进入安装流程，根据回显页面执行**y**或**Y**确认协议，输入其他任意字符为拒绝协议，确认接受协议后开始安装。

若当前语言环境不满足要求，可以执行如下命令配置系统的默认语言环境。

```
#配置为中文（简体）
export LANG=zh_CN.UTF-8
#配置为英文
export LANG=en_US.UTF-8
```

安装完成后，若显示如下信息，则说明软件安装成功：

```
xxx install success
```

xxx表示安装的实际软件包名。

----结束

6.5.2 安装二进制算子包

在包含动态shape网络或单算子API（例如[aclnn](#)类API）场景下需要安装算子包`Ascend-cann-kernels-xxx_linux.run`，支持多个芯片类型的二进制算子包安装在同一台设备上，请通过[6.1 准备软件包](#)章节获取。

二进制算子包依赖CANN开发套件包`Ascend-cann-toolkit_xxx.run`，安装时确保已安装配套版本的toolkit，并使用同一用户安装。请参照[6.5.1 安装开发套件包](#)的步骤安装二进制算子包。

安装完成后所在路径如下（以默认安装路径为例），用户可以在该路径下查询是否已安装二进制算子包：

- root用户：“`/usr/local/Ascend/ascend-toolkit/latest/opp/built-in/op_impl/ai_core/tbe/kernel`”
- 非root用户：“`${HOME}/Ascend/ascend-toolkit/latest/opp/built-in/op_impl/ai_core/tbe/kernel`”

其中\${HOME}为当前用户目录。

📖 说明

若环境中已同时安装了配套版本的toolkit或nnae软件包，安装时需通过以下两种方式识别具体的安装目录。

- 方式一：执行命令时需要添加参数 `--type=<package_type>`，识别到具体的安装目录。
- 方式二：先配置需依赖的toolkit或nnae的环境变量，再执行安装，具体操作请参考[6.5.3 配置环境变量](#)。

6.5.3 配置环境变量

CANN软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

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

用户也可以通过修改`~/.bashrc`文件方式设置永久环境变量，操作如下：

1. 以运行用户在任意目录下执行`vi ~/.bashrc`命令，打开`.bashrc`文件，在文件最后一行后面添加上述内容。
2. 执行`wq!`命令保存文件并退出。
3. 执行`source ~/.bashrc`命令使其立即生效。

6.5.4 安装深度学习框架

6.5.4.1 安装 TensorFlow

6.5.4.1.1 安装开源框架 TensorFlow

若用户仅进行离线推理，请跳过此章节。

安装前准备

- 对于x86架构，跳过安装前准备。
- 对于aarch64架构。

由于TensorFlow依赖h5py，而h5py依赖HDF5，需要先编译安装HDF5，否则使用pip安装h5py会报错，以下步骤以root用户操作。

– 编译安装HDF5。

- i. 访问[下载链接](#)下载HDF5源码包，并上传到安装环境的任意目录。
- ii. 进入源码包所在目录，执行如下命令解压源码包。

```
tar -zxvf hdf5-1.10.5.tar.gz
```
- iii. 进入解压后的文件夹，执行配置、编译和安装命令：

```
cd hdf5-1.10.5/  
./configure --prefix=/usr/include/hdf5  
make  
make install
```
- iv. 配置环境变量并建立动态链接库软连接。

- 1) 配置环境变量。

```
export CPATH="/usr/include/hdf5/include:/usr/include/hdf5/lib/"
```
- 2) root用户建立动态链接库软连接命令如下，非root用户需要在以下命令前添加sudo。

```
ln -s /usr/include/hdf5/lib/libhdf5.so /usr/lib/libhdf5.so
ln -s /usr/include/hdf5/lib/libhdf5_hl.so /usr/lib/libhdf5_hl.so
```

- 安装h5py。

root用户下执行如下命令安装h5py依赖包。

```
pip3 install Cython==0.29
```

root用户下执行如下命令安装h5py，TensorFlow 1.15需安装h5py 2.8.0版本，TensorFlow 2.6.5需安装h5py 3.1.0版本，请所需版本修改如下命令。

```
pip3 install h5py==2.8.0
```

安装 TensorFlow

说明

- TensorFlow1.15配套的Python版本是：Python3.7.x（3.7.5~3.7.11）。
- TensorFlow2.6.5配套的Python版本是：Python3.7.x（3.7.5~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.2）。
- TensorFlow2.6.5存在漏洞，请参考[相关漏洞及其修复方案](#)处理。

需要安装TensorFlow才可以进行算子开发验证、训练业务开发。

- 对于x86架构：直接从pip源下载即可，系统要求等具体请参考[TensorFlow官网](#)。需要注意TensorFlow官网提供的指导描述有误，从pip源下载CPU版本需要显式指定tensorflow-cpu，如果不指定CPU，默认下载的是GPU版本。安装命令参考如下：

如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install tensorflow-cpu==1.15 --user**

- 安装TensorFlow1.15

```
pip3 install tensorflow-cpu==1.15
```

- 安装TensorFlow2.6.5

```
pip3 install tensorflow-cpu==2.6.5
```

- 对于aarch64架构：由于pip源未提供对应的版本，所以需要用户使用官网要求的linux_gcc7.3.0编译器编译tensorflow1.15.0或tensorflow2.6.5，编译步骤参考官网[TensorFlow官网](#)。特别注意点如下。

在下载完tensorflow tag v1.15.0或tensorflow tag v2.6.5源码后需要执行如下步骤。

步骤1 下载“nsync-1.22.0.tar.gz”源码包。

- 进入源码目录，TensorFlow1.15场景下打开“tensorflow/workspace.bzl”文件，TensorFlow2.6.5场景下打开“tensorflow/workspace2.bzl”文件，找到其中name为nsync的“tf_http_archive”定义。

```
tf_http_archive(
    name = "nsync",
    sha256 = "caf32e6b3d478b78cff6c2ba009c3400f8251f646804bcb65465666a9cea93c4",
    strip_prefix = "nsync-1.22.0",
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),
    urls = [
        "https://storage.googleapis.com/mirror.tensorflow.org/github.com/google/nsync/archive/1.22.0.tar.gz",
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",
    ],
)
```

- 从urls中的任一路径下载nsync-1.22.0.tar.gz的源码包，保存到任意路径。

步骤2 修改“nsync-1.22.0.tar.gz”源码包。

- 切换到nsync-1.22.0.tar.gz所在路径，解压缩该源码包。解压缩后存在“nsync-1.22.0”文件夹。

2. 编辑“nsync-1.22.0/platform/c++11/atomic.h”。

在NSYNC_CPP_START_内容后添加如下加粗字体内容。

```
#include "nsync_cpp.h"
#include "nsync_atomic.h"

NSYNC_CPP_START_

#define ATM_CB_() __sync_synchronize()

static INLINE int atm_cas_nomb_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
&o, n, std::memory_order_relaxed, std::memory_order_relaxed));
    ATM_CB_();
    return result;
}

static INLINE int atm_cas_acq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
&o, n, std::memory_order_acquire, std::memory_order_relaxed));
    ATM_CB_();
    return result;
}

static INLINE int atm_cas_rel_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
&o, n, std::memory_order_release, std::memory_order_relaxed));
    ATM_CB_();
    return result;
}

static INLINE int atm_cas_relacq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
&o, n, std::memory_order_acq_rel, std::memory_order_relaxed));
    ATM_CB_();
    return result;
}
```

步骤3 重新压缩“nsync-1.22.0.tar.gz”源码包。

将上个步骤中解压出的内容压缩为一个新的“nsync-1.22.0.tar.gz”源码包，保存（比如，保存在“/tmp/nsync-1.22.0.tar.gz”）。

步骤4 重新生成“nsync-1.22.0.tar.gz”源码包的sha256sum校验码。

执行如下命令后得到sha256sum校验码（一串数字和字母的组合）。

```
sha256sum /tmp/nsync-1.22.0.tar.gz
```

步骤5 修改sha256sum校验码和urls。

进入tensorflow tag源码目录，TensorFlow1.15场景下打开“tensorflow/workspace.bzl”文件，TensorFlow2.6.5场景下打开“tensorflow/workspace2.bzl”文件，找到其中name为nsync的“tf_http_archive”定义，其中“sha256=”后面的数字填写步骤4得到的校验码，“urls=”后面的列表第二行，填写存放“nsync-1.22.0.tar.gz”的file://索引。

```
tf_http_archive(
    name = "nsync",
    sha256 = "caf32e6b3d478b78c62ba009c3400f8251f646804bcb65465666a9cea93c4",
    strip_prefix = "nsync-1.22.0",
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),
    urls = [
        "https://storage.googleapis.com/mirror.tensorflow.org/github.com/google/nsync/archive/1.22.0.tar.gz",
        "file:///tmp/nsync-1.22.0.tar.gz ",
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",
    ],
)
```

步骤6 继续参考官网的“配置build”（[TensorFlow官网](#)）章节执行编译。

执行完./configure之后，需要修改 .tf_configure.bazelrc 配置文件。

添加如下一行build编译选项，ABI可以配置为0或者1，如下以配置为0举例：

```
build:opt --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0
```

删除以下两行：

```
build:opt --copt=-march=native  
build:opt --host_copt=-march=native
```

步骤7 继续执行官方的编译指导步骤（[TensorFlow官网](#)）即可。

步骤8 安装编译好的TensorFlow。

以上步骤执行完后会打包TensorFlow到指定目录，进入指定目录后执行如下命令安装：

如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install tensorflow-1.15.0-*.whl --user**

- TensorFlow1.15:
pip3 install tensorflow-1.15.0-*.whl
- TensorFlow2.6.5:
pip3 install tensorflow-2.6.5-*.whl

步骤9 执行如下命令验证安装效果。

```
python3 -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

如果返回了张量则表示安装成功。

----结束

6.5.4.1.2 安装框架插件包

在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装框架插件包。包括如下两种方式：

方式一：不定制，请直接使用run包安装

1. 获取Ascend-cann-tfplugin_xxx.run。
2. 请参照[6.5.1 安装开发套件包](#)的步骤安装框架插件包。软件包安装详细日志路径如下。
 - root用户：“/var/log/ascend_seclog/ascend_tfplugin_install.log”
 - 非root用户：“\${HOME}/var/log/ascend_seclog/ascend_tfplugin_install.log”其中\${HOME}为当前用户目录。

3. 配置环境变量。

CANN软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

```
./usr/local/Ascend/tfplugin/set_env.sh
```

用户也可以通过修改~/.bashrc文件方式设置永久环境变量，操作如下：

- a. 以运行用户在任意目录下执行**vi ~/.bashrc**命令，打开.bashrc文件，在文件最后一行后面添加上述内容。

- b. 执行**wq!**命令保存文件并退出。
- c. 执行**source ~/.bashrc**命令使其立即生效。

方式二：需要定制，请通过源码包方式编译安装

- 1. 下载框架插件源码包。示例命令如下：
`git clone -b tfa_v0.0.23_8.0.RC1.alpha003 https://gitee.com/ascend/tensorflow.git`
- 2. 参见下载包中的“README.md”进行修改、编译和安装。

6.5.4.2 安装 PyTorch

6.5.4.2.1 安装前必读

为使用PyTorch框架的开发者提供昇腾AI处理器的超强算力，昇腾开发PyTorch Ascend Adapter插件（torch_npu）用于适配PyTorch框架，本章节指导用户在昇腾环境安装PyTorch框架和torch_npu插件。

若用户仅进行离线推理，请跳过此章节。

前提条件

- 安装配套版本的CANN软件并配置CANN环境变量。
- 安装二进制算子包具体请参考6.5.2 安装二进制算子包。
- 安装对应框架版本的torchvision，PyTorch 1.11.0需安装0.12.0版本，PyTorch 2.0.1版本需安装0.15.2版本，PyTorch 2.1.0版本需安装0.16.0版本。
执行如下命令安装torchvision，以0.12.0版本为例：
`pip3 install torchvision==0.12.0`
- 通过源码编译安装torch_npu插件时，安装如下环境依赖。
执行如下命令安装。如果使用非root用户安装，需要在命令后加--user，例如：
`pip3 install pyyaml --user`
`pip3 install pyyaml`
`pip3 install wheel`
`pip3 install setuptools`

版本配套关系

表 6-6 PyTorch 版本配套关系

CANN版本	PyTorch版本	PyTorch Ascend Adapter 插件版本	PyTorch Ascend Adapter 插件代码分支名称	PyTorch Ascend Adapter 插件安装包版本	Python版本	AscendHub 镜像版本/名称
7.0.0.alpha003	PyTorch1.11.0	5.0.0	v1.11.0-5.0.0	1.11.0.post8	Python3.7.x（3.7.5及以上）、Python3.8.x、Python3.9.x、Python3.10.x	-

CANN版本	PyTorch版本	PyTorch Ascend Adapter 插件版本	PyTorch Ascend Adapter 插件代码分支名称	PyTorch Ascend Adapter 插件安装包版本	Python版本	AscendHub 镜像版本/名称
	PyTorch2.0.1		v2.0.1-5.0.0	2.0.1.post1	Python3.8.x、Python3.9.x、Python3.10.x	-
	PyTorch2.1.0		v2.1.0-5.0.0	2.1.0	Python3.8.x、Python3.9.x、Python3.10.x	-

安装步骤

须知

安全提示：建议使用非管理员账户安装运行torch_npu，且建议对安装程序的目录文件做好权限管控，建议文件夹权限设置为750，文件权限设置为 640。可以通过设置umask控制安装后文件的权限，如设置umask为0027。更多安全相关内容请参考[6.5.4.2.5 安全说明](#)。

在昇腾环境安装PyTorch步骤如下：

- 1. 安装PyTorch框架。
- 2. 安装torch_npu插件：可以通过源码编译安装。
- 3. （可选）安装APEX模块：混合精度训练时安装。

说明

- 如果用户不确定自己的软件包版本，可参考[A.8 查询软件包版本信息](#)进行查询。

6.5.4.2.2 安装 PyTorch 框架

x86_64 架构

说明

在x86_64架构下，官方torch包使用MKL加速库。如果需要使用其他blas和lapack加速库（如openblas）来提升性能，请使用源码编译安装方式安装官方torch包，步骤请参考[PyTorch官方链接](#)。

方式一：

使用以下命令可以直接从PyTorch官方预编译包的下载页面中，选择与本地环境匹配的版本包来安装PyTorch，包括匹配操作系统平台（例如Linux）和匹配系统的Python版本号（例如cp37）。例如torch==1.11.0+cpu：指定安装torch版本为1.11.0，并且是CPU版本。

- 安装1.11.0版本
pip3 install torch==1.11.0+cpu -f https://download.pytorch.org/whl/torch_stable.html

- 安装2.0.1版本
pip3 install torch==2.0.1+cpu -f https://download.pytorch.org/whl/torch_stable.html
- 安装2.1.0版本
pip3 install torch==2.1.0+cpu -f https://download.pytorch.org/whl/torch_stable.html

方式二：

步骤1 点击下方PyTorch官方链接下载whl包。PyTorch安装包版本需与Python版本一一对应，例如Python版本为3.8.x，需下载"torch-*version*+cpu-**cp38-cp38-linux_x86_64**.whl"PyTorch安装包。更多PyTorch版本可访问<https://download.pytorch.org/whl/torch/>查询。

参数说明：

- torch：表示PyTorch。
- version：PyTorch的版本号，例如1.11.0。
- cpu：说明是CPU版本的PyTorch。
- cp38-cp38：适用于Python 3.8的版本。
- linux_x86_64：适用于linux系统的x86_64架构。
- whl：Python编译好的wheel二进制发布包格式。

表 6-7 x86_64 常用下载链接

PyTorch版本	Python版本	下载链接
1.11.0	Python 3.7和Python 3.7m	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp37-cp37m-linux_x86_64.whl
	Python 3.8	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp39-cp39-linux_x86_64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp310-cp310-linux_x86_64.whl
2.0.1	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp39-cp39-linux_x86_64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp310-cp310-linux_x86_64.whl
2.1.0	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp39-cp39-linux_x86_64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp310-cp310-linux_x86_64.whl

步骤2 执行以下命令安装。以安装适配Python 3.8的PyTorch 1.11.0版本为例，其他版本请根据下载的软件包修改命令：

```
pip3 install torch-1.11.0+cpu-cp38-cp38-linux_x86_64.whl
```

如果使用非root用户安装，需要在命令后加--user。

----结束

aarch64

- 步骤1 获取安装包。更多PyTorch版本可访问<https://download.pytorch.org/whl/torch/>查询。PyTorch安装包版本需与Python版本一一对应，例如Python版本为3.8.x，需下载"torch-version-cp38-cp38-manylinux2014_aarch64.whl"PyTorch安装包。其中：
- torch：表示PyTorch。
 - version：PyTorch的版本号，例如1.11.0。
 - cp38-cp38：表示这个安装包是为Python 3.8编译的。
 - manylinux2014_aarch64：表示支持manylinux2014标准的aarch64架构系统(64位ARM)。
 - whl：表明这是一个Python wheel格式的二进制发布包。

表 6-8 aarch64 常用下载链接

PyTorch版本	Python版本	下载链接
1.11.0	Python 3.7和Python 3.7m	https://download.pytorch.org/whl/torch-1.11.0-cp37-cp37m-manylinux2014_aarch64.whl
	Python 3.8	https://download.pytorch.org/whl/torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/torch-1.11.0-cp39-cp39-manylinux2014_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/torch-1.11.0-cp310-cp310-manylinux2014_aarch64.whl
2.0.1	Python 3.8	https://download.pytorch.org/whl/torch-2.0.1-cp38-cp38-manylinux2014_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/torch-2.0.1-cp39-cp39-manylinux2014_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/torch-2.0.1-cp310-cp310-manylinux2014_aarch64.whl
2.1.0	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp38-cp38-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl

用户可使用上方链接直接下载。或者使用wget命令下载，以下载适配Python 3.8的的PyTorch 1.11.0版本为例，执行如下安装命令（其他版本请根据下载的软件包修改命令）：

```
wget https://download.pytorch.org/whl/torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
```

步骤2 执行如下命令安装。以安装适配Python 3.8的PyTorch 1.11.0版本为例，执行如下安装命令（其他版本请根据下载的软件包修改命令）：

```
pip3 install torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
```

如果使用非root用户安装，需要在命令后加--user。

----结束

6.5.4.2.3 安装 torch_npu 插件

编译安装适用于进行算子适配开发、CANN版本与PyTorch兼容适配场景下。

以下操作步骤以安装PyTorch 2.0.1版本为例。

步骤1 安装依赖。

选择编译安装方式安装时需要安装系统依赖。目前支持CentOS与Ubuntu操作系统。

- CentOS

```
yum install -y patch libjpeg-turbo-devel dos2unix openblas git
yum install -y gcc==7.3.0 cmake==3.12.0
```

- Ubuntu

```
apt-get install -y patch build-essential libbz2-dev libreadline-dev wget curl llvm libncurses5-dev
libncursesw5-dev xz-utils tk-dev liblzma-dev m4 dos2unix libopenblas-dev git
apt-get install -y gcc==7.3.0 cmake==3.12.0
```

gcc7.3.0版本及以上，cmake3.12.0版本及以上；若需要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。

步骤2 编译生成torch_npu插件的二进制安装包。CANN和PyTorch的配套关系请通过[单击这里](#)查看。

1. 下载对应PyTorch版本分支代码，进入插件根目录，以v6.0.rc1.alpha003-pytorch2.0.1为例，其他版本请下载对应PyTorch版本分支代码。

```
git clone -b v6.0.rc1.alpha003-pytorch2.0.1 https://gitee.com/ascend/pytorch.git
cd pytorch
```

2. （可选）如果不需要切换opplugin（CANN）的版本，请跳过此步骤；如需切换opplugin版本，如切换opplugin的 master分支最新代码，则进行以下操作进行切换。

同步submodule，将opplugin等三方库同步到本地。

```
git submodule init && git submodule update
cd third_party/op-plugin/
# 拉取特定版本的代码并切换到该版本，此处以master为例
git fetch origin master && git checkout master
cd ../../
```

3. 编译生成二进制安装包。

```
# 指定Python版本编包方式，以Python3.8为例，其他Python版本请使用 --python=3.9或--python3.10
bash ci/build.sh --python=3.8
```

步骤3 安装pytorch/dist目录下生成的插件torch_npu包，如果使用非root用户安装，需要在命令后加--user。

```
# 请用户根据实际情况更改命令中的torch_npu包名
pip3 install --upgrade dist/torch_npu-2.0.1.post1-cp38-cp38-linux_aarch64.whl
```

步骤4 验证是否安装成功。

```
python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu(); print(a + a);"
```

显示如下回显证明PyTorch框架与插件安装成功。

```
new kernel registered at /home/p21/pytorch_v1.10.1/pytorch
tensor([[ -0.6066,  6.3385,  0.0379,  3.3356],
        [ 2.9243,  3.3134, -1.5465,  0.1916],
        [-2.1807,  0.2008, -1.1431,  2.1523]], device='npu:0')
```

----结束

6.5.4.2.4 安装 APEX 模块

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芯片的架构特性，用户需要开启混合精度，可以提升模型的性能。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。

混合精度的介绍可参考《PyTorch 训练模型迁移调优指南》中的“模型迁移与训练>自动混合精度（AMP）”章节，推荐用户使用框架自带的AMP功能。

APEX模块的使用介绍可参考《PyTorch 训练模型迁移调优指南》中的“模型套件和第三方库>APEX”章节。

推荐用户通过编译源码包安装APEX模块。

步骤1 安装依赖。

选择编译安装方式安装时需要安装系统依赖。目前支持CentOS与Ubuntu操作系统。

- CentOS

```
yum install -y patch libjpeg-turbo-devel dos2unix openblas git
yum install -y gcc==7.3.0 cmake==3.12.0
```
- Ubuntu

```
apt-get install -y patch build-essential libbz2-dev libreadline-dev wget curl llvm libncurses5-dev
libncursesw5-dev xz-utils tk-dev liblzma-dev m4 dos2unix libopenblas-dev git
apt-get install -y gcc==7.3.0 cmake==3.12.0
```

gcc7.3.0版本及以上，cmake3.12.0版本及以上；若需要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。

步骤2 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。

```
pip install setuptools==65.7.0
```

步骤3 获取昇腾适配的APEX源码。

```
git clone -b 5.0.0 https://gitee.com/ascend/apex.git
```

步骤4 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=3.7 # 支持python3.7、3.8、3.9、3.10
```

说明

请确保NPU版本的PyTorch可以正常使用，否则会影响APEX的编译。

步骤5 执行如下命令安装。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python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-cp37-cp37m-linux_aarch64.whl
```

📖 说明

pip版本要求小于等于24.0，以确保APEX正常安装。可参考如下命令进行指定pip版本的安装：

```
python -m pip install pip==version
```

----结束

6.5.4.2.5 安全说明

漏洞提示

PyTorch 1.11.0存在CVE-2022-45907漏洞，该漏洞因torch.jit.annotations.parse_type_line方法中不安全使用eval方法，存在可以执行任意代码风险。torch_npu已参考[LINK](#)进行修复。

安全加固

网络加固

PyTorch分布式训练服务需要在设备间进行通信，通信开启的端口默认为全0侦听，为了避免安全风险，建议用户针对此场景进行安全加固。如使用iptables配置防火墙，在运行分布式训练开始前限制外部对分布式训练使用端口的访问，在运行分布式训练结束后清理防火墙规则。

- 防火墙规则设定，可参考如下脚本：

```
#!/bin/bash
set -x

# 要限制的端口号
port={端口号}

# 清除旧规则
iptables -D INPUT -p tcp -j {规则名}
iptables -F {规则名}
iptables -X {规则名}

# 创建新的规则链
iptables -t filter -N {规则名}

# 在单机场景下设定白名单，允许其他节点访问主节点的侦听端口
# 在规则链中添加允许特定IP地址范围的规则
iptables -t filter -A {规则名} -i eno1 -p tcp --dport $port -s {允许外部访问的IP} -j ACCEPT

# 屏蔽外部地址访问分布式训练端口
# 在PORT-LIMIT-RULE规则链中添加拒绝其他IP地址的规则
iptables -t filter -A {规则名} -i {要限制的网卡名} -p tcp --dport $port -j DROP

# 将流量传递给规则链
iptables -I INPUT -p tcp -j {规则名}
```

- 防火墙规则移除，可参考如下脚本：

```
#!/bin/bash
set -x

# 清除规则
iptables -D INPUT -p tcp -j {规则名}
iptables -F {规则名}
iptables -X {规则名}
```

防火墙规则设定和移除示例：

1. 针对特定端口设定防火墙，脚本中端口号为要限制的端口，在PyTorch分布式训练中端口号请参考[表3 通信矩阵](#)；要限制的网卡名为服务器用于分布式通信使用的

网卡，允许的外部访问的IP为分布式训练服务器的IP地址。网卡和服务器IP可以通过ifconfig查看，如下文回显的eth0为网卡名，192.168.1.1为服务器IP地址：

```
# ifconfig
eth0
    inet addr:192.168.1.1 Bcast:192.168.1.255 Mask:255.255.255.0
    inet6 addr: fe80::230:64ee:ef1a:c1a/64 Scope:Link
```

2. 假定服务器主节点地址192.168.1.1，另一台需要进行分布式训练的服务器为192.168.1.2，训练端口为29510。

- 规则设定可参考如下脚本：

```
#!/bin/bash
set -x

# 设定侦听的端口
port=29510

# 清除旧规则
iptables -D INPUT -p tcp -j PORT-LIMIT-RULE
iptables -F PORT-LIMIT-RULE
iptables -X PORT-LIMIT-RULE

# 创建新的PORT-LIMIT-RULE规则链
iptables -t filter -N PORT-LIMIT-RULE

# 在多机场景下设定白名单，允许192.168.1.2访问主节点
# 在PORT-LIMIT-RULE规则链中添加允许特定IP地址范围的规则
iptables -t filter -A PORT-LIMIT-RULE -i eno1 -p tcp --dport $port -s 192.168.1.2 -j ACCEPT

# 屏蔽外部地址访问分布式训练端口
# 在PORT-LIMIT-RULE规则链中添加拒绝其他IP地址的规则
iptables -t filter -A PORT-LIMIT-RULE -i eth0 -p tcp --dport $port -j DROP

# 将流量传递给PORT-LIMIT-RULE规则链
iptables -I INPUT -p tcp -j PORT-LIMIT-RULE
```

- 规则移除可参考如下脚本：

```
#!/bin/bash
set -x

# 清除规则
iptables -D INPUT -p tcp -j PORT-LIMIT-RULE
iptables -F PORT-LIMIT-RULE
iptables -X PORT-LIMIT-RULE
```

权限控制

- 建议用户对训练所需文件、训练过程中保存的文件、用户个人的隐私数据、商业资产等敏感文件做好权限控制等安全措施，例如多用户共享数据集场景下的数据集文件写权限控制、profiler等场景产生数据文件权限控制等，设定的权限建议参考[表1 文件权限参考](#)进行设置。
- torch_npu中profiler工具会生成性能记录文件，生成的文件权限为640，文件夹权限为750，用户可根据需要自行对生成后的相关文件进行权限控制。
- 用户安装和使用过程需要做好权限控制，建议参考[表1 文件权限参考](#)进行设置。如需要保存安装/卸载日志，可在安装/卸载命令后面加上参数--log <FILE>，注意对<FILE>文件及目录做好权限管控。
- PyTorch框架运行中所生成的文件权限依赖系统设定，如torch.save接口保存的文件。建议当前执行脚本的用户根据自身需要，对生成文件做好权限控制，设定的权限可参考[表1 文件权限参考](#)。可使用umask控制默认权限，避免提权等安全风险，建议用户将主机（包括宿主机）和容器中的umask设置为027及其以上，提高安全性。

文件权限参考

建议用户根据自身需要，参考[表1 文件权限参考](#)对各类文件进行加固。

表 6-9 文件权限参考

类型	linux权限参考值	备注
文件夹 / 目录	750 (rwxr-x---)	对于共享目录可为755
数据集文件	640 (rw-r-----)	对于共享数据集文件可为644
checkpoint文件	640 (rw-r-----)	对于checkplint等生成文件可以设置为640
程序文件	440 (r--r-----)	除非开发调试场景，正常运行时程序文件不应再次修改
可执行脚本	750 (rwxr-x---)	针对可执行脚本设置750

网络通信

PyTorch提供分布式训练能力，支持在单机和多机场景下进行训练，需要进行网络通信。其中PyTorch需要使用TCP进行通信，torch_npu使用CANN中HCCL在NPU设备间通信，通信端口见[表6-11](#)。用户需要注意并保障节点间通信网络安全，可以使用iptables等方式消减安全风险，可参考[安全加固](#)章节进行网络安全加固。

数据安全

- PyTorch和torch_npu提供的profiler性能分析工具和torch_npu提供AOE性能调优工具都会在本地产生成性能拆解数据。用户需加强对相关数据的保护，需要模型性能调优时使用，调优完成后及时关闭，AOE和Profiler工具具体细节请参考《PyTorch 训练模型迁移调优指南》。
- PyTorch使用过程中需要加载和保存数据，部分接口使用风险模块pickle，可能存在数据风险，如torch.load、torch.distributed.scatter_object_list等接口，可参考[torch.load](#)、[collective-functions](#)了解具体风险。
- AscendPyTorch依赖CANN的基础能力实现AOE性能调优、算子dump、日志记录等功能，用户需关注上述功能生成文件的权限控制。

编译安全

torch_npu支持源码编译安装，在编译时会下载依赖第三方库并执行构建shell脚本，在编译过程中会产生临时程序文件和编译目录。用户可根据需要自行对源代码目录内的文件进行权限管控避免安全风险。

运行提示

- 建议根据运行环境资源状况编写训练脚本。若训练脚本与资源状况不匹配，如数据集的size超出内存容量/NPU存储容量等，可能引发错误并导致进程退出。
- PyTorch和torch_npu在运行异常时会退出进程并打印报错信息，建议根据报错提示定位具体错误原因，包括设定算子同步执行、查看CANN日志、解析生成的Core Dump文件等方式。

公网地址

在torch_npu的配置文件和脚本中存在[表6-10](#)：

表 6-10 公网地址

类型	开源代码地址	文件名	公网IP地址/公网URL地址/域名/邮箱地址	用途说明
自研	不涉及	ci\docker\X86\Dock erfile	https://mirrors.huaweicloud.com/repository/pypi/simple	docker配置文件，用于配置pip源
自研	不涉及	ci\docker\X86\Dock erfile	https://download.pytorch.org/whl/cpu	docker配置源，用于配置torch下载连接
自研	不涉及	ci\docker\ARM\Dock erfile	https://mirrors.aliyun.com/pypi/simple	docker配置文件，用于配置pip源
自研	不涉及	setup.cfg	https://gitee.com/ascend/pytorch	用于打包whl的url入参
自研	不涉及	setup.cfg	https://gitee.com/ascend/pytorch/tags	用于打包whl的download_url入参
自研	不涉及	third_party\op-plugin\ci\build.sh	https://gitee.com/ascend/pytorch.git	编译脚本根据torch_npu仓库地址拉取代码进行编译
自研	不涉及	third_party\op-plugin\ci\exec_ut.sh	https://gitee.com/ascend/pytorch.git	UT脚本根据torch_npu仓库地址下拉取代码进行UT测试
开源引入	https://github.com/pytorch/pytorch/blob/main/test/nn/test_convolution.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_mps.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/legacy_conv2d.pt	用于test脚本下载相关pt文件

类型	开源代码地址	文件名	公网IP地址/公网URL地址/域名/邮箱地址	用途说明
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/legacy_serialized.pt	用于test脚本下载相关pt文件
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/gpu_tensors.pt	用于test脚本下载相关pt文件
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_nn.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/linear.pt	用于test脚本下载相关pt文件

接口说明

AscendPyTorch是PyTorch适配插件，支持用户使用PyTorch在昇腾设备上进行训练和推理。AscendPyTorch适配后支持用户使用PyTorch原生接口。除了原生PyTorch接口外，AscendPyTorch提供了部分自定义接口，包括自定义算子、亲和库和其他接口，支持PyTorch接口和自定义接口连接，具体可参考《PyTorch 训练模型迁移调优指南》>API列表。

参考[PyTorch社区公开接口规范](#)，PyTorchAdapter提供了对外的自定义接口。如果一个函数看起来符合公开接口的标准且在文档中有展示，则该接口是公开接口。否则，使用该功能前可以在社区询问该功能是否确实是公开的或意外暴露的接口，因为这些未暴露接口将来可能会被修改或者删除。

AscendPyTorch使用monkey-patch方式实现PyTorch接口的适配，通过将torch_npu中的部分接口动态替换PyTorch中的接口实现对部分接口的支持。推荐用户使用《PyTorch 训练模型迁移调优指南》>API列表中推荐的调用方式。

AscendPyTorch项目采用C++和Python联合开发，当前除Libtorch场景外正式接口只提供Python接口，在torch_npu的二进制包中动态库不直接提供服务，暴露的接口为内部使用，不建议用户使用。

通信矩阵

通信矩阵具体可参考下表。

表 6-11 通信矩阵

组件	PyTorch	HCCL
源设备	运行torch_npu进程的服务器	运行torch_npu进程的服务器
源IP	设备地址IP	设备地址IP
源端口	操作系统自动分配，分配范围由操作系统的自身配置决定	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
目的设备	运行torch_npu进程的服务器	运行torch_npu进程的服务器
目的IP	设备地址IP	设备地址IP
目的端口（侦听）	默认29500/29400，用户可以设定端口号	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
协议	tcp	tcp

端口说明	1. 在静态分布式场景中（用torchrun/torch.distributed.launch使用的backend为static），目的端口（默认29500）用于接收和发送数据，源端口用于接收和发送数据 2. 在动态分布式场景中（用torchrun或者torch.distributed.launch使用的backend为c10d），目的端口（默认29400）用于接收和发送数据，源端口用于接收和发送数据。可以使用rdzv_endpoint和master_port指定端口号 3. 在分布式场景中，用torchrun或者torch.distributed.launch不指定任何参数时使用的端口号为29500	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
侦听端口是否可更改	是	是
认证方式	无认证方式	无认证方式
加密方式	无	无
所属平面	不涉及	不涉及
版本	所有版本	所有版本
特殊场景	无	无
备注	该通信过程由开源软件PyTorch控制，配置为PyTorch原生设置，可参考 PyTorch文档 。源端口由操作系统自动分配，分配范围由操作系统的配置决定，例如ubuntu：采用/proc/sys/net/ipv4/ipv4_local_port_range文件指定，可通过cat /proc/sys/net/ipv4/ipv4_local_port_range或sysctl net.ipv4.ip_local_port_range查看	该通信过程由CANN中HCCL组件控制，torch_npu不进行控制，端口范围可参考《环境变量参考》的“执行相关 > 集合通信与分布式训练 > 集合通信相关配置 > HCCL_IF_BASE_PORT”和《集合通信接口参考》

6.5.4.3 安装昇思 MindSpore

若用户仅进行离线推理，请跳过此章节。

用户要使用MindSpore框架，请登录[MindSpore官网](#)获取安装MindSpore框架的方法。

6.5.5 安装 Python 版本的 proto

如果训练脚本依赖protobuf的Python版本进行序列化结构的数据存储（例如TensorFlow的序列化相关接口），则需要安装Python版本的proto。

步骤1 检查系统中是否存在“/usr/local/python3.7.5/lib/python3.7/site-packages/google/protobuf/pyext/_message.cpython-37m-*<arch>*-linux-gnu.so”（以参考[6.3 安装依赖](#)章节安装的python3.7.5为例）这个动态库，如果没有，需要按照如下步骤安装。其中*<arch>*为系统架构类型。

说明

“/usr/local/python3.7.5/lib/python3.7/site-packages”是pip安装第三方库的路径，可以使用**pip3 -V**检查。

如果系统显示：/usr/local/python3.7.5/lib/python3.7/site-packages/pip，则pip安装第三方库的路径为/usr/local/python3.7.5/lib/python3.7/site-packages。

步骤2 执行如下命令卸载protobuf。

```
pip3 uninstall protobuf
```

步骤3 下载protobuf软件包。

从<https://github.com/protocolbuffers/protobuf/releases/download/v3.11.3/protobuf-python-3.11.3.tar.gz>路径下下载3.11.3版本protobuf-python-3.11.3.tar.gz软件包（或者其他版本，保证和当前环境上安装的tensorflow兼容），并以root用户上传到所在Linux服务器任意目录并执行命令解压。

```
tar zxvf protobuf-python-3.11.3.tar.gz
```

步骤4 以root用户安装protobuf。

进入protobuf软件包目录。

1. 安装protobuf的依赖。

当操作系统为Ubuntu时，安装命令如下：

```
apt-get install autoconf automake libtool curl make g++ unzip libffi-dev -y
```

当操作系统为CentOS/BClinux时，安装命令如下：

```
yum install autoconf automake libtool curl make gcc-c++ unzip libffi-devel -y
```

2. 为autogen.sh脚本添加可执行权限并执行此脚本。

```
chmod +x autogen.sh
./autogen.sh
```

3. 配置安装路径（默认安装路径为“/usr/local”）。

```
./configure
```

如果想指定安装路径，可参考以下命令。

```
./configure --prefix=/protobuf
```

“/protobuf”为用户指定的安装路径。

4. 执行protobuf安装命令。

```
make -j15      # 通过grep -w processor /proc/cpuinfo|wc -l查看cpu数，示例为15，用户可自行设置相应参数。
make install
```

5. 刷新共享库。

```
ldconfig
```

protobuf安装完成后，会在[步骤4.3](#)中配置的路径下面的include目录中生成google/protobuf文件夹，存放protobuf相关头文件；在[步骤4.3](#)中配置路径下面的bin目录中生成protoc可执行文件，用于进行*.proto文件的编译，生成protobuf的C++头文件及实现文件。

6. 检查是否安装完成。

```
ln -s /protobuf/bin/protoc /usr/bin/protoc
protoc --version
```

其中/**protobuf**为[步骤4.3](#)中用户配置的安装路径。如果用户未配置安装路径，则直接执行**protoc --version**检查是否安装成功。

步骤5 安装protobuf的python版本运行库。

1. 进入protobuf软件包目录的python子目录，编译python版本的运行库。

```
python3 setup.py build --cpp_implementation
```

说明

这里需要编译二进制版本的运行库，如果使用**python3 setup.py build**命令无法生成二进制版本的运行库，在序列化结构的处理时性能会非常慢。

2. 安装动态库。

```
cd .. && make install
```

进入python子目录，安装python版本的运行库。

```
python3 setup.py install --cpp_implementation
```

3. 检查是否安装成功。

检查系统中是否存在“/usr/local/python3.7.5/lib/python3.7/site-packages/protobuf-3.11.3-py3.7-linux-aarch64.egg/google/protobuf/pyext/_message.cpython-37m-*<arch>*-linux-gnu.so”这个动态库。其中*<arch>*为系统架构类型。

说明

“/usr/local/python3.7.5/lib/python3.7/site-packages”是pip安装第三方库的路径，可以使用**pip3 -V**检查。

如果系统显示：/usr/local/python3.7.5/lib/python3.7/site-packages/pip，则pip安装第三方库的路径为/usr/local/python3.7.5/lib/python3.7/site-packages。

4. 若用户在[步骤4.3](#)中指定了安装路径，则需在运行脚本中增加环境变量的设置：

```
export LD_LIBRARY_PATH=/protobuf/lib:${LD_LIBRARY_PATH}
```

“/protobuf”为[步骤4.3](#)中用户配置的安装路径。

5. 建立软连接。

当用户自行配置安装路径时，需要建立软连接，否则导入tensorflow会报错。命令如下：

```
ln -s /protobuf/lib/libprotobuf.so.22.0.3 /usr/lib/libprotobuf.so.22
```

其中“/protobuf”为[步骤4.3](#)中用户配置的安装路径。

----结束

6.5.6 配置 device 的网卡 IP

当进行分布式训练时，需要通过昇腾软件中的HCCN Tool工具配置device的网卡IP，用于多个device间通信以实现网络模型参数的同步更新。如果用户需要使用HCCN Tool工

具的其他功能（如检查网口Link状态），请参见《[Atlas 中心训练服务器 23.0.x\(23.0.3及其系列版本\) HCCN Tool 接口参考](#)》或《[Atlas A2 中心推理和训练硬件 23.0.x\(23.0.3及其系列版本\) HCCN Tool 接口参考](#)》。

Atlas 800 训练服务器、Atlas 900 AI 集群、Atlas 300T 训练卡

Atlas 800 训练服务器（型号 9000）、Atlas 800 训练服务器（型号 9010）、Atlas 900 AI集群（型号 9000）、Atlas 300T 训练卡（型号 9000）、Atlas 300T Pro 训练卡（型号：9000）请参考《[Ascend Training Solution 23.0.RC1 组网指南](#)》的“参数面网络介绍>中大规模集群组网/小规模组网>参数面网络配置示例”章节中“配置训练节点”，通过HCCN_Tool配置NPU网口的device IP。

Atlas A2 训练系列产品

Atlas 800T A2 训练服务器、Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元、Atlas 200T A2 Box16 异构子框请参考《[Ascend Training Solution 23.0.0 组网指南](#)》的“参数面网络配置介绍>配置示例（选择对应的产品）>配置训练节点”，通过HCCN_Tool配置NPU网口的device IP。

7 安装运行环境（nnrt 软件，在物理机/虚拟机安装）

[7.1 安装前必读](#)

[7.2 准备软件包](#)

[7.3 准备安装及运行用户](#)

[7.4 安装离线推理引擎包](#)

[7.5 配置环境变量](#)

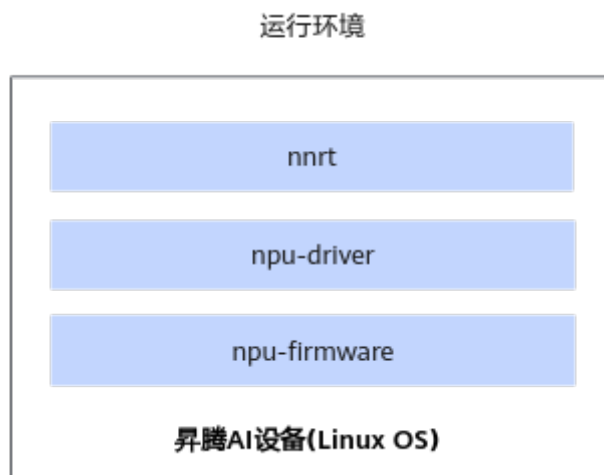
7.1 安装前必读

参考本章节安装运行环境，仅支持离线推理。

运行环境上会运行用户开发的应用程序。

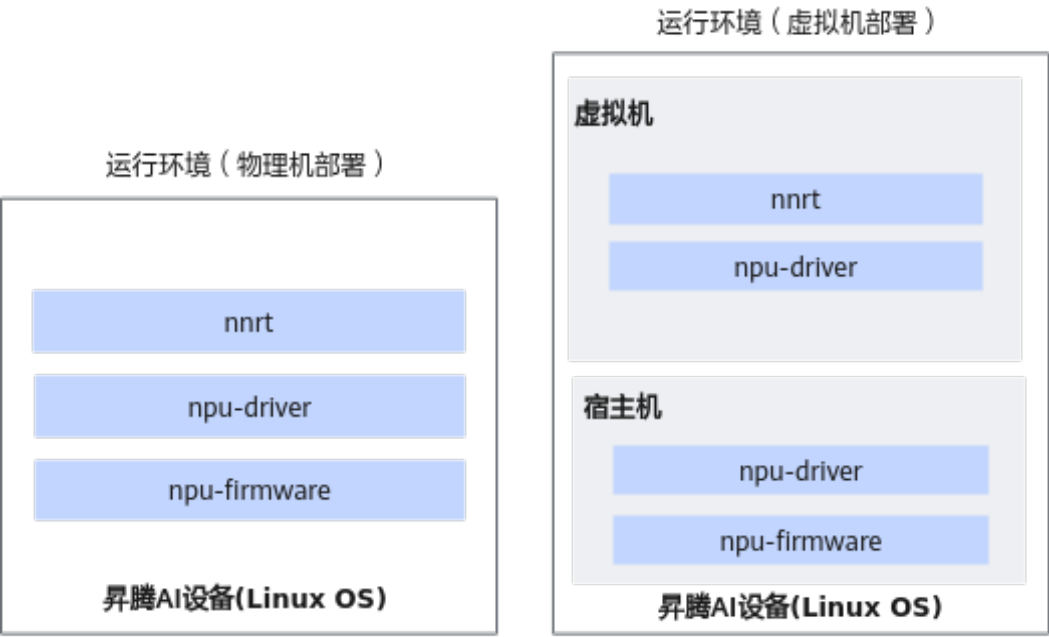
- Atlas 200I A2 加速模块A200I A2 加速模块（RC场景）和Atlas 500 A2 智能小站A500 A2 智能小站安装软件如[图7-1](#)所示。

图 7-1 运行环境



- Atlas 200 AI加速模块（RC场景）运行环境安装请参考《[Atlas 200 AI加速模块 6.0.0 软件安装与维护指南（RC场景）](#)》。
- 若配置的昇腾AI设备为Atlas 200 AI加速模块（EP场景），则应用程序运行环境无容器部署场景。
- 其他设备的运行环境（仅支持离线推理）安装软件如图7-2所示。用户可进行物理机、虚拟机部署。

图 7-2 运行环境



7.2 准备软件包

下载软件包

用户可根据下表获取所需软件包，下载本软件即表示您同意[华为企业业务最终用户许可协议（EULA）](#)的条款和条件。

表 7-1 软件包

软件包类型	软件包名称	说明	获取链接
离线推理引擎包	Ascend-cann-nnrt_{version}_linux-{arch}.run	仅支持离线推理，主要包含AscendCL库、编译依赖的相关库（不包含driver包中的库），用于应用程序的模型推理。	获取链接

说明

{version}表示软件版本号，{arch}表示CPU架构。

软件数字签名验证

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的数字签名文件用于完整性验证。

在软件包下载之后，请参考《OpenPGP签名验证指南》，对下载的软件包进行PGP数字签名校验。如果校验失败，请不要使用该软件包，先联系华为技术支持工程师解决。

使用软件包安装/升级之前，也需要按上述过程先验证软件包的数字签名，确保软件包未被篡改。

运营商客户请访问：<http://support.huawei.com/carrier/digitalSignatureAction>

企业客户请访问：<https://support.huawei.com/enterprise/zh/tool/pgp-verify-TL1000000054>

7.3 准备安装及运行用户

- 运行用户：实际运行推理业务的用户。
- 安装用户：实际安装软件包的用户。
 - 若使用root用户安装，支持所有用户运行相关业务。
 - 若使用非root用户安装，则安装及运行用户必须相同。
 - 已有非root用户，则无需再次创建。
 - 若想使用新的非root用户，则需要先创建该用户，请参见如下方法创建。

说明

- 如果安装驱动时未携带“--install-for-all”，并且CANN软件包运行用户为非root，则该CANN软件包运行用户所属的属组必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 运行用户不建议为root用户属组，权限控制可能存在安全风险，请谨慎使用。

创建非root用户操作方法如下，如下命令请以root用户执行。

1. 创建非root用户。

```
groupadd usergroup  
useradd -g usergroup -d /home/username -m username -s /bin/bash
```

2. 设置非root用户密码。

```
passwd username
```

说明

- 创建完运行用户后，请勿关闭该用户的登录认证功能。
- 设置的口令需符合口令复杂度要求（请参见[C.4 口令复杂度要求](#)）。密码有效期为90天，您可以在/etc/login.defs文件中修改有效期的天数，或者通过chage命令来设置用户的有效期限，详情请参见[A.6 设置用户有效期](#)。

7.4 安装离线推理引擎包

前提条件

- 请参见[7.3 准备安装及运行用户](#)完成安装前准备。
- 在安装软件前请确保安装环境已安装推理卡或AI加速模块的驱动和固件。
- 通过[7.2 准备软件包](#)章节获取离线推理引擎包Ascend-cann-nnrt_{version}_linux-{arch}.run。

安装步骤

获取并安装离线推理引擎包，详细安装步骤如下。

步骤1 以软件包的安装用户登录安装环境。

步骤2 将获取到的离线推理引擎包上传到安装环境任意路径（如“/home/package”）。

步骤3 进入软件包所在路径。

步骤4 执行如下命令增加对软件包的可执行权限。

```
chmod +x 软件包名.run
```

软件包名.run表示软件包名，例如离线推理引擎包Ascend-cann-nnrt_{version}_linux-{arch}.run。请根据实际包名进行替换。

步骤5 执行如下命令校验软件包安装文件的一致性和完整性。

```
./软件包名.run --check
```

步骤6 执行如下命令安装软件（以下命令支持--install-path=<path>等参数，具体参数说明请参见[C.1 参数说明](#)）。

```
./软件包名.run --install
```

说明

- 离线推理引擎包支持不同用户在同一运行环境安装，但安装版本必须保持一致，不同用户所属的属组也必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 用户想使用默认签署[华为企业业务最终用户许可协议（EULA）](#)的方式安装软件包时，可以添加--quiet参数配合安装命令使用，如：./软件包名.run --install --quiet 添加该参数后会跳过[步骤7](#)的确认操作。
- 如果以root用户安装，**不允许安装在非root用户目录下**。
- 如果用户未指定安装路径，则软件会安装到默认路径下，默认安装路径如下。
 - root用户：“/usr/local/Ascend”
 - 非root用户：“\${HOME}/Ascend”其中\${HOME}为当前用户目录。软件包支持多版本安装，同一安装路径下，可以安装多个版本的软件包，不同版本的软件通过版本目录来区分。默认使用最后安装的版本（即通过软件包名/latest建立软连接到最后安装的版本）。
- 软件包安装详细日志路径如下。
 - root用户：“/var/log/ascend_seclog/ascend_nnrt_install.log”
 - 非root用户：“\${HOME}/var/log/ascend_seclog/ascend_nnrt_install.log”其中\${HOME}为当前用户目录。

步骤7 用户需签署[华为企业业务最终用户许可协议（EULA）](#)后进入安装流程，根据回显页面执行**y或Y**确认协议，输入其他任意字符为拒绝协议，确认接受协议后开始安装。

若当前语言环境不满足要求，可以执行如下命令配置系统的默认语言环境。

```
#配置为中文（简体）
export LANG=zh_CN.UTF-8
#配置为英文
export LANG=en_US.UTF-8
```

安装完成后，若显示如下信息，则说明软件安装成功：

```
xxx install success
```

xxx表示安装的实际软件包名。

----结束

添加业务进程到 Cgroup

针对Atlas 200I A2 加速模块A200I A2 加速模块（RC场景）和Atlas 500 A2 智能小站A500 A2 智能小站场景，提供了基于cgroup（Control Groups）机制的内存限制功能，当业务进程的cgroup达到特定的内存限额后，内核会尝试回收内存。添加方法如下：

在启动过程中通过**getpid()** 函数获取到当前进程的pid，再将其写入**/sys/fs/cgroup/memory/usermemory/cgroup.procs**文件中。示例代码如下，加粗部分是关键代码。

```
int write_task(void)
{
    int pid = getpid();
    int fd = -1;
    char buf[100] = {0};
    int ret;
    fd = open("/sys/fs/cgroup/memory/usermemory/cgroup.procs", O_RDWR, S_IRUSR | S_IWUSR);
    if (fd < 0) {
        printf("can't open pidfile\n");
        return (-1);
    }
    ret = sprintf(buf, "%ld", (long)getpid());
    if (ret < 0) {
        printf("sprintf_s failed.");
        close(fd);
        return (1);
    }
    ret = write(fd, buf, strlen(buf) + 1);
    if (ret != (strlen(buf) + 1)) {
        printf("write failed, errno.\n");
        close(fd);
        return (1);
    }
    close(fd);
    return 0;
}
```

7.5 配置环境变量

nnrt等软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

```
# 安装nnrt包时配置
./usr/local/Ascend/nnrt/set_env.sh
```

用户也可以通过修改**~/.bashrc**文件方式设置永久环境变量，操作如下：

1. 以运行用户在任意目录下执行`vi ~/.bashrc`命令，打开`.bashrc`文件，在文件最后一行后面添加上述内容。
2. 执行`wq!`命令保存文件并退出。
3. 执行`source ~/.bashrc`命令使其立即生效。

8 安装运行环境（nnae 软件，在物理机/虚拟机安装）

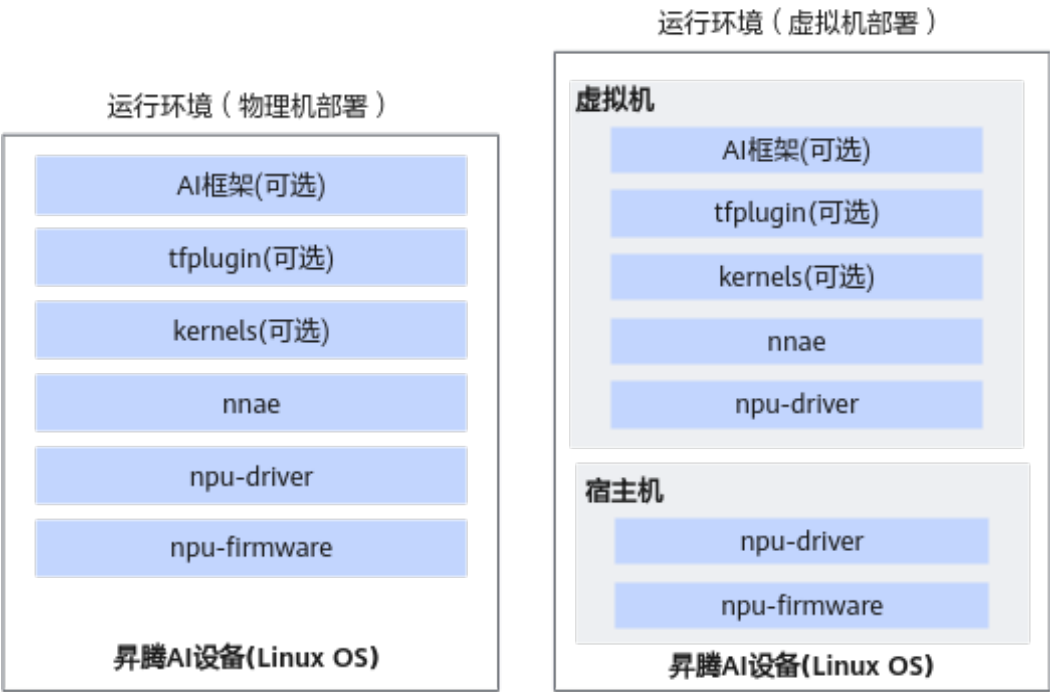
- 8.1 安装前必读
- 8.2 准备软件包
- 8.3 准备安装及运行用户
- 8.4 安装依赖
- 8.5 安装深度学习引擎包
- 8.6 安装二进制算子包
- 8.7 配置环境变量
- 8.8 安装深度学习框架
- 8.9 安装Python版本的proto
- 8.10 配置device的网卡IP

8.1 安装前必读

参考本章节安装运行环境，支持离线推理、在线推理和训练。

运行环境安装软件如[图8-1](#)所示。用户可进行物理机和虚拟机部署。

图 8-1 运行环境



8.2 准备软件包

下载软件包

用户可根据下表获取所需软件包，下载本软件即表示您同意[华为企业业务最终用户许可协议（EULA）](#)的条款和条件。

表 8-1 软件包

名称	软件包	说明	获取链接
深度学习引擎包	Ascend-cann-nnae_{version}_linux-{arch}.run	- 支持离线推理、在线推理、训练，包含FWK库Fwklib和算子库OPP组件。 - 请根据CPU架构（x86_64、aarch64）获取对应的软件包。	获取链接
二进制算子包	Ascend-cann-kernels-{chip_type}_{version}_linux.run	（可选）二进制算子包，可以节省算子编译时间。 在包含动态shape网络或单算子API（例如aclnn类API）场景下，需要获取该软件包。	获取链接

名称	软件包	说明	获取链接
框架插件包	Ascend-cann-tfplugin_{version}_linux-{arch}.run	插件包，对接上层框架TensorFlow的适配插件。 在线推理或训练场景下若使用深度学习框架TensorFlow，需要获取该软件包。	获取链接

说明

{version}表示软件版本号，{arch}表示CPU架构，{chip_type}表示芯片类型，可在昇腾AI处理器执行`npu-smi info`查看。

软件数字签名验证

为了防止软件包在传递过程或存储期间被恶意篡改，下载软件包时需下载对应的数字签名文件用于完整性验证。

在软件包下载之后，请参考《OpenPGP签名验证指南》，对下载的软件包进行PGP数字签名校验。如果校验失败，请不要使用该软件包，先联系华为技术支持工程师解决。

使用软件包安装/升级之前，也需要按上述过程先验证软件包的数字签名，确保软件包未被篡改。

运营商客户请访问：<http://support.huawei.com/carrier/digitalSignatureAction>

企业客户请访问：<https://support.huawei.com/enterprise/zh/tool/pgp-verify-TL1000000054>

8.3 准备安装及运行用户

- 运行用户：实际运行推理业务或执行训练的用户。
- 安装用户：实际安装软件包的用户。
 - 若使用root用户安装，支持所有用户运行相关业务。
 - 若使用非root用户安装，则安装及运行用户必须相同。
 - 已有非root用户，则无需再次创建。
 - 若想使用新的非root用户，则需要先创建该用户，请参见如下方法创建。

说明

- 如果安装驱动时未携带“--install-for-all”，并且CANN软件包运行用户为非root，则该CANN软件包运行用户所属的属组必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 运行用户不建议为root用户属组，权限控制可能存在安全风险，请谨慎使用。

创建非root用户操作方法如下，如下命令请以root用户执行。

- 创建非root用户。

```
groupadd usergroup
useradd -g usergroup -d /home/username -m username -s /bin/bash
```

2. 设置非root用户密码。

```
passwd username
```

 说明

- 创建完运行用户后， 请勿关闭该用户的登录认证功能。
- 设置的口令需符合口令复杂度要求（请参见[C.4 口令复杂度要求](#)）。密码有效期为90天，您可以在/etc/login.defs文件中修改有效期的天数， 或者通过chage命令来设置用户的有效期限，详情请参见[A.6 设置用户有效期](#)。

8.4 安装依赖

8.4.1 安装前必读

安装CANN软件前需安装相关依赖。

本章节以Ubuntu 22.04、openEuler 22.03和SLES 12.5为例，详述依赖安装操作。其他系统可参考这三种系统进行安装。

- Ubuntu、Debian、UOS20、UOS20 SP1、Linux系统可参考Ubuntu 22.04进行安装。
- EulerOS、openEuler、BCLinux、BCLinux-for-Euler、CentOS、CTyunOS、Kylin、UOS20 1020e、Ant OS、CULinux系统可参考openEuler 22.03进行安装。
- SLES系统可参考SLES 12.5进行安装。

8.4.2 依赖列表

 说明

针对用户自行安装的开源软件，如Python、numpy等，请使用稳定版本（尽量使用无漏洞的版本）。

Ubuntu 22.04

表 8-2 依赖信息

名称	版本要求
Python	CANN支持Python3.7.x（3.7.0~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.7）、Python3.10.x（3.10.0~3.10.12），其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 • TensorFlow1.15支持Python3.7.x（3.7.0~3.7.11）。 • Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x（3.7.5~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.2）。

名称	版本要求
cmake	要求3.5.1版本及以上cmake，若cmake版本低于3.5.1，请参见 A.3 安装3.5.2版本cmake 解决。
make	-
gcc	要求7.3.0版本及以上gcc，若gcc版本低于7.3.0，可参考 A.4 安装7.3.0版本gcc 进行安装。
g++	
zlib1g zlib1g-dev libsqlite3-dev openssl libssl-dev libffi-dev libbz2-dev libxslt1-dev unzip pciutils net-tools libblas-dev gfortran libblas3 liblapack-dev liblapack3 libopenblas-dev	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.19.2
decorator	>=4.4.0
sympy	>=1.5.1
cffi	>=1.12.3
protobuf	>=3.13.0
attrs pyyaml pathlib2 scipy requests psutil absl-py	无版本要求，安装的版本以pip源为准。

openEuler 22.03

表 8-3 依赖信息

名称	版本限制
Python	CANN支持Python3.7.x (3.7.0~3.7.11)、Python3.8.x (3.8.0~3.8.11)、Python3.9.x (3.9.0~3.9.7)、Python3.10.x (3.10.0~3.10.12)，其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 - TensorFlow1.15支持Python3.7.x (3.7.0~3.7.11)。 - Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x (3.7.5~3.7.11)、Python3.8.x (3.8.0~3.8.11)、Python3.9.x (3.9.0~3.9.2)。
cmake	要求3.5.1版本及以上cmake，若cmake版本低于3.5.1，请参见 A.3 安装3.5.2版本cmake 解决。
make	-
gcc	要求7.3.0版本及以上gcc，若gcc版本低于7.3.0，可参考 A.4 安装7.3.0版本gcc 进行安装。
g++	
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools sqlite-devel lapack-devel gcc-gfortran python3-devel	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.19.2
decorator	>=4.4.0
sympy	>=1.4
cffi	>=1.12.3
protobuf	>=3.13.0

名称	版本限制
attrs pyyaml pathlib2 scipy requests psutil absl-py	无版本要求，安装的版本以pip源为准。

Suse 12SP5

表 8-4 依赖信息

类别	版本限制
Python	CANN支持Python3.7.x（3.7.0~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.7）、Python3.10.x（3.10.0~3.10.12），其中Ascend-cann-tfplugin和当前适配的TensorFlow不支持Python3.10.x。 说明 - TensorFlow1.15支持Python3.7.x（3.7.0~3.7.11）。 - Ascend-cann-tfplugin和TensorFlow2.6.5框架支持Python3.7.x（3.7.5~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.2）。
cmake	>=3.5.1
make	-
gcc	离线推理场景： 要求4.8.5版本及以上gcc。
gcc-c++	
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools gdbm-devel	无版本要求，安装的版本以操作系统自带的源为准。
numpy	>=1.14.3
decorator	>=4.4.0
sympy	>=1.4

类别	版本限制
cffi	>=1.12.3
protobuf	>=3.11.3
attrs pyyaml pathlib2 scipy requests psutil gnureadline absl-py	无版本要求，安装的版本以pip源为准。

8.4.3 安装步骤（Ubuntu 22.04）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
apt-get update
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，则检查网络是否连接或者把“/etc/apt/sources.list”文件中的源更换为可用的源或使用镜像源（以配置华为镜像源为例，可参考[华为开源镜像站](#)）。

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```
3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。
 - a. 在任意目录下执行如下命令，打开.bashrc文件：

```
vi ~/.bashrc
```

在文件最后一行后面添加umask 0022内容。
 - b. 执行:wq!命令保存文件并退出。
 - c. 执行source ~/.bashrc命令使其立即生效。

说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
gcc --version
g++ --version
make --version
cmake --version
dpkg -l zlib1g| grep zlib1g| grep ii
dpkg -l zlib1g-dev| grep zlib1g-dev| grep ii
dpkg -l libsqlite3-dev| grep libsqlite3-dev| grep ii
dpkg -l openssl| grep openssl| grep ii
dpkg -l libssl-dev| grep libssl-dev| grep ii
dpkg -l libffi-dev| grep libffi-dev| grep ii
dpkg -l libbz2-dev| grep libbz2-dev| grep ii
dpkg -l libxslt1-dev| grep libxslt1-dev| grep ii
dpkg -l unzip| grep unzip| grep ii
dpkg -l pciutils| grep pciutils| grep ii
dpkg -l net-tools| grep net-tools| grep ii
dpkg -l libblas-dev| grep libblas-dev| grep ii
dpkg -l gfortran| grep gfortran| grep ii
dpkg -l libblas3| grep libblas3| grep ii
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[8.4.2 依赖列表](#)为准）。

```
gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
g++ (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
GNU Make 4.3
cmake version 3.22.1
ii  zlib1g:arm64 1:1.2.11.dfsg-2ubuntu9 arm64      compression library - runtime
ii  zlib1g-dev:arm64 1:1.2.11.dfsg-2ubuntu9 arm64      compression library - development
ii  libsqlite3-dev:arm64 3.37.2-2 arm64      SQLite 3 development files
ii  openssl 3.0.2-0ubuntu1 arm64      Secure Sockets Layer toolkit - cryptographic utility
ii  libssl-dev:arm64 3.0.2-0ubuntu1 arm64      Secure Sockets Layer toolkit - development files
ii  libffi-dev:arm64 3.4.2-4 arm64      Foreign Function Interface library (development files)
ii  libbz2-dev:amd64 1.0.6-8.1ubuntu0.2 arm64      high-quality block-sorting file compressor library - development
ii  libxslt1-dev:amd64 1.1.29-5ubuntu0.3 arm64      XSLT 1.0 processing library - development kit
ii  unzip 6.0-26ubuntu3 arm64      De-archiver for .zip files
ii  pciutils 1:3.7.0-6 arm64      PCI utilities
ii  net-tools 1.60+git20181103.0eebece-1ubuntu5 arm64      NET-3 networking toolkit
libblas-dev:arm64 3.9.0-1build1 arm64      Basic Linear Algebra Subroutines 3, static library
gfortran 4:9.3.0-1ubuntu2 arm64      GNU Fortran 95 compiler
libblas3:arm64 3.9.0-1build1 arm64      Basic Linear Algebra Reference implementations, shared library
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将步骤1至步骤2命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行su - *username*命令切换到非root用户继续执行步骤1至步骤3。
- libsqlite3-dev需要在python安装之前安装，如果用户操作系统已经安装满足版本要求的python环境，在此之后再安装libsqlite3-dev，则需要重新编译python环境。

```
sudo apt-get install -y gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev libbz2-dev libxslt1-dev unzip pciutils net-tools libblas-dev gfortran libblas3
```

步骤2 检查系统是否安装满足版本要求的python开发环境，具体要求请参见[8.4.2 依赖列表](#)。

执行如下命令，如果返回信息满足python版本要求，则直接进入下一步，否则可参考[A.2 编译安装Python3.7.5](#)。

```
python3 --version
pip3 --version
```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```
pip3 install attrs
pip3 install numpy
pip3 install decorator
pip3 install sympy
pip3 install cffi
pip3 install pyyaml
pip3 install pathlib2
pip3 install psutil
pip3 install protobuf
pip3 install scipy
pip3 install requests
pip3 install absl-py
```

如果执行上述命令时报错“subprocess.CalledProcessError: Command '('lsb_release', '-a')' return non-zero exit status 1”，请参见[B.6 pip3 install报错“subprocess.CalledProcessError: Command '\('lsb_release', '-a'\)' return non-zero exit status 1”](#)。

----结束

说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

8.4.4 安装步骤（openEuler 22.03）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
yum makecache
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，则检查网络是否连接或者把“/etc/yum.repos.d/xxxx.repo”文件中的源更换为可用的源或使用镜像源（以配置华为镜像源为例，可参考[华为开源镜像站](#)）。

说明

如果执行上述命令提示“Your license is invalid”，请获取OS授权license。

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```
3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。
 - a. 在任意目录下执行如下命令，打开**.bashrc**文件：

```
vi ~/.bashrc
```


在文件最后一行后面添加umask 0022内容。
 - b. 执行**wq!**命令保存文件并退出。
 - c. 执行**source ~/.bashrc**命令使其立即生效。

说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中**umask 0022**一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

配置最大线程数

训练场景下，不同OS的最大线程数可能不满足训练要求，需执行以下命令修改最大线程数为无限制。

1. 以root用户登录安装环境。
2. 配置环境变量，修改线程数为无限制，编辑“/etc/profile”文件，在文件的最后添加如下内容后保存退出：

```
ulimit -u unlimited
```
3. 执行如下命令使环境变量生效。

```
source /etc/profile
```

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
gcc --version
g++ --version
make --version
cmake --version
rpm -qa |grep unzip
rpm -qa |grep zlib-devel
rpm -qa |grep libffi-devel
rpm -qa |grep openssl-devel
rpm -qa |grep pciutils
rpm -qa |grep net-tools
rpm -qa |grep sqlite-devel
rpm -qa |grep lapack-devel
rpm -qa |grep gcc-gfortran
rpm -qa |grep python3-devel
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[8.4.2 依赖列表](#)为准）。

```
gcc (GCC) 10.3.1
g++ (GCC) 10.3.1
GNU Make 4.3
cmake version 3.22.0
unzip-6.0-45.oe1.aarch64
zlib-devel-1.2.11-22.oe1.aarch64
libffi-devel-3.4.2-2.oe2203.aarch64
openssl-devel-1.1.1m-18.oe2203.aarch64
pciutils-3.7.0-2.oe2203.aarch64
net-tools-2.10-3.oe2203.aarch64
sqlite-devel-3.24.0-9.oe1.aarch64
lapack-devel-3.8.0-16.oe1.aarch64
gcc-gfortran-7.3.0-20190804.h31.oe1.aarch64
python3-devel-3.7.4-8.oe1.aarch64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将**步骤1至步骤2**命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行**su - username**命令切换到非root用户继续执行**步骤1至步骤3**。
- sqlite-devel需要在python安装之前安装，如果用户操作系统已经安装满足版本要求的python环境，在此之后再安装sqlite-devel，则需要重新编译python环境。

```
sudo yum install -y gcc gcc-c++ make cmake unzip zlib-devel libffi-devel openssl-devel pciutils net-tools
sqlite-devel lapack-devel gcc-gfortran python3-devel
```

如果通过上述方式安装的cmake版本低于3.5.1，则请参见[A.3 安装3.5.2版本cmake](#)解决。

步骤2 检查系统是否安装满足版本要求的python开发环境，具体要求请参见[8.4.2 依赖列表](#)。

执行如下命令，如果返回信息满足python版本要求，则直接进入下一步，否则可参考[A.2 编译安装Python3.7.5](#)。

```
python3 --version
pip3 --version
```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```
pip3 install attrs
pip3 install numpy
pip3 install decorator
pip3 install sympy
pip3 install cffi
pip3 install pyyaml
pip3 install pathlib2
pip3 install psutil
pip3 install protobuf
pip3 install scipy
pip3 install requests
pip3 install absl-py
```

- 如果安装numpy报错，请参考[B.5 pip3 install numpy报错](#)解决。
- 如果安装scipy报错，请参考[B.4 pip3 install scipy报错](#)解决。

步骤4 如果仅需支持离线推理，请跳过此步骤。

CentOS7.6系统使用软件源默认安装的gcc版本为4.8.5，因此需要安装7.3.0版本gcc，安装过程请参见[A.4 安装7.3.0版本gcc](#)。

----结束

说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

8.4.5 安装步骤（SLES 12.5）

检查源

安装过程需要下载相关依赖，请确保安装环境能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
zypper ref
```

如果命令执行报错或者后续安装依赖时等待时间过长甚至报错，可参考以下操作配置本地源。

请以root用户执行如下操作。

1. 挂载系统镜像iso文件（以下命令为示例）。

```
mount -o loop SLE-12-SP5-Server-DVD-x86_64-GM-DVD1.iso /mnt
```
2. 使用ar参数添加一个mnt路径下挂好的本地源，并命名为suse（用户可自定义）。

```
zypper ar -f /mnt suse
```
3. 清空zypper缓存，刷新源，使配置的源生效。

```
zypper clean  
zypper ref
```

检查 root 用户的 umask

1. 以root用户登录安装环境。
2. 检查root用户的umask值。

```
umask
```
3. 如果umask不等于0022，请执行如下操作配置，在该文件的最后一行添加umask 0022后保存。
 - a. 在任意目录下执行如下命令，打开.bashrc文件：

```
vi ~/.bashrc
```


在文件最后一行后面添加umask 0022内容。
 - b. 执行:wq!命令保存文件并退出。
 - c. 执行source ~/.bashrc命令使其立即生效。

说明

依赖安装完成后，请用户恢复为原umask值（删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

配置安装用户权限

可使用root或非root用户（该非root用户需与软件包安装用户保持一致）安装依赖，如果使用非root用户安装，可能需要用到提权命令，请用户自行获取所需的sudo权限。使用完成后请取消涉及高危命令的权限，否则有sudo提权风险。

安装依赖

步骤1 检查系统是否安装python依赖以及gcc等软件。

分别使用如下命令检查是否安装gcc，make以及python依赖软件等。

```
rpm -qa | grep gcc
rpm -qa | grep make
rpm -qa | grep unzip
rpm -qa | grep zlib-devel
rpm -qa | grep openssl-devel
rpm -qa | grep pciutils
rpm -qa | grep net-tools
rpm -qa | grep gdbm-devel
rpm -qa | grep libffi-devel
```

若分别返回如下信息则说明已经安装，进入下一步（以下回显仅为示例，版本要求请以[8.4.2 依赖列表](#)为准）。

```
gcc48-4.8.5-31.20.1.x86_64
libgcc_s1-8.2.1+r264010-1.3.3.x86_64
gcc-4.8-6.189.x86_64
libgcc_s1-32bit-8.2.1+r264010-1.3.3.x86_64
gcc-c++-4.8-6.189.x86_64
gcc48-c++-4.8.5-31.20.1.x86_64
cmake-3.5.2-20.6.1.x86_64
makedumpfile-1.6.5-1.19.x86_64
automake-1.13.4-6.2.noarch
make-4.0-4.1.x86_64
unzip-6.00-33.8.1.x86_64
zlib-devel-1.2.11-3.21.1.x86_64
zlib-devel-32bit-1.2.11-3.21.1.x86_64
zlib-devel-static-1.2.11-3.21.1.x86_64
zlib-devel-1.2.11-9.42.x86_64
zlib-devel-static-32bit-1.2.11-3.21.1.x86_64
libopenssl-devel-1.0.2p-1.13.noarch
pciutils-3.2.1-11.3.1.x86_64
pciutils-ids-2018.02.08-12.3.1.noarch
net-tools-1.60-765.5.4.x86_64
gdbm-devel-1.10-9.70.x86_64
libffi-devel-3.2.1.git259-10.8.x86_64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为还未安装的软件即可）：

说明

- 如果使用root用户安装依赖，请将步骤1至步骤2命令中的sudo删除。
- 如果python及其依赖是使用非root用户安装，则需要执行su - *username*命令切换到非root用户继续执行步骤1至步骤3。

```
sudo zypper install -y gcc gcc-c++ make cmake unzip zlib-devel openssl-devel pciutils net-tools gdbm-devel
```

由于本地源中缺少libffi-devel依赖，可从[opensuse镜像源](#)中下载libffi-devel-3.2.1.git259-10.8.x86_64.rpm、libffi7-3.2.1.git259-10.8.x86_64.rpm、libffi7-32bit-3.2.1.git259-10.8.x86_64.rpm（软件包会定时更新，请以实际rpm包名为准）并一同上传至服务器某一目录下（如“/home/test”）。

进入rpm包所在路径（如“/home/test”），执行如下命令安装所需依赖：

```
sudo rpm -ivh *.rpm --nodeps
```

步骤2 检查系统是否安装满足版本要求的python开发环境（具体要求请参见[8.4.2 依赖列表](#)，此步骤以环境上需要使用python 3.7.x为例进行说明）。

执行命令**python3 --version**，如果返回信息满足python版本要求，则直接进入下一步。

否则可参考如下方式安装python3.7.5。

1. 使用wget下载python3.7.5源码包，可以下载到安装环境的任意目录，命令为：
`wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz`

2. 进入下载后的目录，解压源码包，命令为：
`tar -zxvf Python-3.7.5.tgz`

3. 进入解压后的文件夹，创建安装目录，执行配置、编译和安装命令：
`cd Python-3.7.5`
`./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared`
`make`
`sudo make install`

其中“--prefix”参数用于指定python安装路径，用户根据实际情况进行修改，“--enable-shared”参数用于编译出libpython3.7m.so.1.0动态库。

本手册以--prefix=/usr/local/python3.7.5路径为例进行说明。执行配置、编译和安装命令后，安装包在/usr/local/python3.7.5路径，libpython3.7m.so.1.0动态库在/usr/local/python3.7.5/lib/libpython3.7m.so.1.0路径。

4. 设置python3.7.5环境变量。
#用于设置python3.7.5库文件路径
`export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH`
#如果用户环境存在多个python3版本，则指定使用python3.7.5版本
`export PATH=/usr/local/python3.7.5/bin:$PATH`

通过以上export方式设置环境变量，该种方式设置的环境变量只在当前窗口有效。您也可以通过将以上命令写入~/.bashrc文件中，然后执行**source ~/.bashrc**命令，使上述环境变量永久生效。注意如果后续您有使用环境上其他python版本的需求，则不建议将以上命令写入到~/.bashrc文件中。

5. 安装完成之后，执行如下命令查看安装版本，如果返回相关版本信息，则说明安装成功。

```
python3 --version  
pip3 --version
```

步骤3 安装前请先使用**pip3 list**命令检查是否安装相关依赖，若已经安装，则请跳过该步骤；若未安装，则安装命令如下（如果只有部分软件未安装，则如下命令修改为只安装还未安装的软件即可）。

- 请在安装前配置好pip源，具体可参考[A.7 配置pip源](#)。
- 安装前，建议执行命令**pip3 install --upgrade pip**进行升级，避免因pip版本过低导致安装失败。
- 如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install attrs --user**，安装命令可在任意路径下执行。

```
pip3 install attrs  
pip3 install numpy  
pip3 install decorator  
pip3 install sympy  
pip3 install cffi  
pip3 install pyyaml  
pip3 install pathlib2  
pip3 install psutil  
pip3 install protobuf  
pip3 install scipy  
pip3 install requests
```

```
pip3 install gnureadline
pip3 install absl-py
```

----结束

说明

依赖安装完成后，请用户恢复为原umask值（参考[检查root用户的umask](#)，删除.bashrc文件中umask 0022一行）。基于安全考虑，建议用户将umask值改为0027。

8.5 安装深度学习引擎包

前提条件

- 在安装软件前请确保安装环境已安装NPU驱动。
- 通过[8.2 准备软件包](#)章节获取深度学习引擎包Ascend-cann-nnae_{version}_linux-{arch}.run。

安装步骤

获取并安装深度学习引擎包，详细安装步骤如下。

步骤1 以软件包的安装用户登录安装环境。

若[8.4 安装依赖](#)中安装依赖的用户为root用户，则软件包的安装用户可自行指定；若[8.4 安装依赖](#)中安装依赖的用户为非root用户，请确保软件包的安装用户与该用户保持一致。

步骤2 将深度学习引擎包上传到安装环境任意路径（如“/home”）。

步骤3 进入软件包所在路径。

步骤4 执行如下命令增加对软件包的可执行权限。

```
chmod +x 软件包名.run
```

软件包名.run表示软件包名，例如深度学习引擎包Ascend-cann-nnae_{version}_linux-{arch}.run。请根据实际包名进行替换。

步骤5 执行如下命令校验软件包安装文件的一致性和完整性。

```
./软件包名.run --check
```

步骤6 执行如下命令安装软件（以下命令支持--install-path=<path>等参数，具体参数说明请参见[C.1 参数说明](#)）。

```
./软件包名.run --install
```

说明

- 深度学习引擎包支持不同用户在同一运行环境安装，但安装版本必须保持一致，不同用户所属的属组也必须和驱动运行用户所属属组相同；如果不同，请用户自行添加到驱动运行用户属组。
- 用户想使用默认签署[华为企业业务最终用户许可协议（EULA）](#)的方式安装软件包时，可以添加`--quiet`参数配合安装命令使用，如：`./软件包名.run --install --quiet` 添加该参数后会跳过[步骤7](#)的确认操作。
- 如果以root用户安装，**不允许安装在非root用户目录下**。
- 如果用户未指定安装路径，则软件会安装到默认路径下，默认安装路径如下。
 - root用户：“`/usr/local/Ascend`”
 - 非root用户：“`${HOME}/Ascend`”其中`${HOME}`为当前用户目录。软件包支持多版本安装，同一安装路径下，可以安装多个版本的软件包，不同版本的软件通过版本目录来区分。默认使用最后安装的版本（即通过[软件包名/latest](#)建立软链接到最后安装的版本）。
- 软件包安装详细日志路径如下。
 - root用户：“`/var/log/ascend_seclog/ascend_nnae_install.log`”
 - 非root用户：“`${HOME}/var/log/ascend_seclog/ascend_nnae_install.log`”其中`${HOME}`为当前用户目录。

步骤7 用户需签署[华为企业业务最终用户许可协议（EULA）](#)后进入安装流程，根据回显页面执行`y`或`Y`确认协议，输入其他任意字符为拒绝协议，确认接受协议后开始安装。

若当前语言环境不满足要求，可以执行如下命令配置系统的默认语言环境。

```
#配置为中文（简体）
export LANG=zh_CN.UTF-8
#配置为英文
export LANG=en_US.UTF-8
```

安装完成后，若显示如下信息，则说明软件安装成功：

```
xxx install success
```

xxx表示安装的实际软件包名。

----结束

8.6 安装二进制算子包

在包含动态shape网络或单算子API（例如aclnn类API）场景下需要安装算子包Ascend-cann-kernels-xxx_linux.run，支持多个芯片类型的二进制算子包安装在同一台设备上，请通过[8.2 准备软件包](#)章节获取。

二进制算子包依赖CANN深度学习引擎包Ascend-cann-nnae_xxx.run，安装时确保已安装配套版本的nnae，并使用同一用户安装。请参照[8.5 安装深度学习引擎包](#)的步骤安装二进制算子包。

安装完成后所在路径如下（以默认安装路径为例），用户可以在该路径下查询是否已安装二进制算子包：

- root用户：“`/usr/local/Ascend/nnae/latest/opp/built-in/op_impl/ai_core/tbe/kernel`”
- 非root用户：“`${HOME}/Ascend/nnae/latest/opp/built-in/op_impl/ai_core/tbe/kernel`”

其中`${HOME}`为当前用户目录。

说明

若环境中已同时安装了配套版本的toolkit或nnae软件包，安装时需通过以下两种方式识别具体的安装目录。

- 方式一：执行命令时需要添加参数 `--type=<package_type>`，识别到具体的安装目录。
- 方式二：先配置需依赖的toolkit或nnae的环境变量，再执行安装，具体操作请参考[8.7 配置环境变量](#)。

8.7 配置环境变量

nnae等软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

```
# 安装nnae包时配置
./usr/local/Ascend/nnae/set_env.sh
```

用户也可以通过修改`~/.bashrc`文件方式设置永久环境变量，操作如下：

1. 以运行用户在任意目录下执行`vi ~/.bashrc`命令，打开`.bashrc`文件，在文件最后一行后面添加上述内容。
2. 执行`wq!`命令保存文件并退出。
3. 执行`source ~/.bashrc`命令使其立即生效。

8.8 安装深度学习框架

8.8.1 安装 TensorFlow

8.8.1.1 安装开源框架 TensorFlow

若用户仅进行离线推理，请跳过此章节。

安装前准备

- 对于x86架构，跳过安装前准备。
- 对于aarch64架构。

由于TensorFlow依赖h5py，而h5py依赖HDF5，需要先编译安装HDF5，否则使用pip安装h5py会报错，以下步骤以root用户操作。

– 编译安装HDF5。

i. 访问[下载链接](#)下载HDF5源码包，并上传到安装环境的任意目录。

ii. 进入源码包所在目录，执行如下命令解压源码包。

```
tar -zxvf hdf5-1.10.5.tar.gz
```

iii. 进入解压后的文件夹，执行配置、编译和安装命令：

```
cd hdf5-1.10.5/
./configure --prefix=/usr/include/hdf5
make
make install
```

iv. 配置环境变量并建立动态链接库软连接。

1) 配置环境变量。

```
export CPATH="/usr/include/hdf5/include/:/usr/include/hdf5/lib/"
```

- 2) root用户建立动态链接库软连接命令如下，非root用户需要在以下命令前添加sudo。

```
ln -s /usr/include/hdf5/lib/libhdf5.so /usr/lib/libhdf5.so
ln -s /usr/include/hdf5/lib/libhdf5_hl.so /usr/lib/libhdf5_hl.so
```

- 安装h5py。

root用户下执行如下命令安装h5py依赖包。

```
pip3 install Cython==0.29
```

root用户下执行如下命令安装h5py，TensorFlow 1.15需安装h5py 2.8.0版本，TensorFlow 2.6.5需安装h5py 3.1.0版本，请所需版本修改如下命令。

```
pip3 install h5py==2.8.0
```

安装 TensorFlow

说明

- TensorFlow1.15配套的Python版本是：Python3.7.x（3.7.5~3.7.11）。
- TensorFlow2.6.5配套的Python版本是：Python3.7.x（3.7.5~3.7.11）、Python3.8.x（3.8.0~3.8.11）、Python3.9.x（3.9.0~3.9.2）。
- TensorFlow2.6.5存在漏洞，请参考[相关漏洞及其修复方案](#)处理。

需要安装TensorFlow才可以进行算子开发验证、训练业务开发。

- 对于x86架构：直接从pip源下载即可，系统要求等具体请参考[TensorFlow官网](#)。需要注意TensorFlow官网提供的指导描述有误，从pip源下载CPU版本需要显式指定tensorflow-cpu，如果不指定CPU，默认下载的是GPU版本。安装命令参考如下：

如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install tensorflow-cpu==1.15 --user**

- 安装TensorFlow1.15

```
pip3 install tensorflow-cpu==1.15
```
- 安装TensorFlow2.6.5

```
pip3 install tensorflow-cpu==2.6.5
```

- 对于aarch64架构：由于pip源未提供对应的版本，所以需要用户使用官网要求的linux_gcc7.3.0编译器编译tensorflow1.15.0或tensorflow2.6.5，编译步骤参考官网[TensorFlow官网](#)。特别注意点如下。

在下载完tensorflow tag v1.15.0或tensorflow tag v2.6.5源码后需要执行如下步骤。

步骤1 下载“nsync-1.22.0.tar.gz”源码包。

1. 进入源码目录，TensorFlow1.15场景下打开“tensorflow/workspace.bzl”文件，TensorFlow2.6.5场景下打开“tensorflow/workspace2.bzl”文件，找到其中name为nsync的“tf_http_archive”定义。

```
tf_http_archive(
    name = "nsync",
    sha256 = "caf32e6b3d478b78cff6c2ba009c3400f8251f646804bcb65465666a9cea93c4",
    strip_prefix = "nsync-1.22.0",
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),
    urls = [
        "https://storage.googleapis.com/mirror.tensorflow.org/github.com/google/nsync/archive/1.22.0.tar.gz",
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",
    ],
)
```

2. 从urls中的任一路径下载nsync-1.22.0.tar.gz的源码包，保存到任意路径。

步骤2 修改“nsync-1.22.0.tar.gz”源码包。

1. 切换到nsync-1.22.0.tar.gz所在路径，解压缩该源码包。解压缩后存在“nsync-1.22.0”文件夹。
2. 编辑“nsync-1.22.0/platform/c++11/atomic.h”。

在NSYNC_CPP_START_内容后添加如下加粗字体内容。

```
#include "nsync_cpp.h"
#include "nsync_atomic.h"

NSYNC_CPP_START_

#define ATM_CB() __sync_synchronize()

static INLINE int atm_cas_nomb_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
    &o, n, std::memory_order_relaxed, std::memory_order_relaxed));
    ATM_CB();
    return result;
}

static INLINE int atm_cas_acq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
    &o, n, std::memory_order_acquire, std::memory_order_relaxed));
    ATM_CB();
    return result;
}

static INLINE int atm_cas_rel_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
    &o, n, std::memory_order_release, std::memory_order_relaxed));
    ATM_CB();
    return result;
}

static INLINE int atm_cas_relacq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),
    &o, n, std::memory_order_acq_rel, std::memory_order_relaxed));
    ATM_CB();
    return result;
}
```

步骤3 重新压缩“nsync-1.22.0.tar.gz”源码包。

将上个步骤中解压出的内容压缩为一个新的“nsync-1.22.0.tar.gz”源码包，保存（比如，保存在“/tmp/nsync-1.22.0.tar.gz”）。

步骤4 重新生成“nsync-1.22.0.tar.gz”源码包的sha256sum校验码。

执行如下命令后得到sha256sum校验码（一串数字和字母的组合）。

```
sha256sum /tmp/nsync-1.22.0.tar.gz
```

步骤5 修改sha256sum校验码和urls。

进入tensorflow tag源码目录，TensorFlow1.15场景下打开“tensorflow/workspace.bzl”文件，TensorFlow2.6.5场景下打开“tensorflow/workspace2.bzl”文件，找到其中name为nsync的“tf_http_archive”定义，其中“sha256=”后面的数字填写**步骤4**得到的校验码，“urls=”后面的列表第二行，填写存放“nsync-1.22.0.tar.gz”的file://索引。

```
tf_http_archive(
    name = "nsync",
    sha256 = "caf32e6b3d478b78cff6c2ba009c3400f8251f646804bcb65465666a9cea93c4",
    strip_prefix = "nsync-1.22.0",
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),
    urls = [
        "https://storage.googleapis.com/mirror.tensorflow.org/github.com/google/nsync/archive/1.22.0.tar.gz",
        "file:///tmp/nsync-1.22.0.tar.gz ",
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",
```

```
l,  
)
```

步骤6 继续参考官网的“配置build”（[TensorFlow官网](#)）章节执行编译。

执行完./configure之后，需要修改 .tf_configure.bazelrc 配置文件。

添加如下一行build编译选项，ABI可以配置为0或者1，如下以配置为0举例：

```
build:opt --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0
```

删除以下两行：

```
build:opt --copt=-march=native  
build:opt --host_copt=-march=native
```

步骤7 继续执行官方的编译指导步骤（[TensorFlow官网](#)）即可。

步骤8 安装编译好的TensorFlow。

以上步骤执行完后会打包TensorFlow到指定目录，进入指定目录后执行如下命令安装：

如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install tensorflow-1.15.0-*.whl --user**

- TensorFlow1.15:
pip3 install tensorflow-1.15.0-*.whl
- TensorFlow2.6.5:
pip3 install tensorflow-2.6.5-*.whl

步骤9 执行如下命令验证安装效果。

```
python3 -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

如果返回了张量则表示安装成功。

----结束

8.8.1.2 安装框架插件包

在线推理或训练场景下若使用深度学习框架TensorFlow，需要安装框架插件包。包括如下两种方式：

方式一：不定制，请直接使用run包安装

1. 获取Ascend-cann-tfplugin_xxx.run。
2. 请参照[6.5.1 安装开发套件包](#)的步骤安装框架插件包。软件包安装详细日志路径如下。
 - root用户：“/var/log/ascend_seclog/ascend_tfplugin_install.log”
 - 非root用户：“\${HOME}/var/log/ascend_seclog/ascend_tfplugin_install.log”其中\${HOME}为当前用户目录。
3. 配置环境变量。

CANN软件提供进程级环境变量设置脚本，供用户在进程中引用，以自动完成环境变量设置。用户进程结束后自动失效。示例如下（以root用户默认安装路径为例）：

```
./usr/local/Ascend/tfplugin/set_env.sh
```

用户也可以通过修改`~/.bashrc`文件方式设置永久环境变量，操作如下：

- 以运行用户在任意目录下执行`vi ~/.bashrc`命令，打开`.bashrc`文件，在文件最后一行后面添加上述内容。
- 执行`wq!`命令保存文件并退出。
- 执行`source ~/.bashrc`命令使其立即生效。

方式二：需要定制，请通过源码包方式编译安装

1. [下载](#)框架插件源码包。示例命令如下：

```
git clone -b tfa_v0.0.23_8.0.RC1.alpha003 https://gitee.com/ascend/tensorflow.git
```

2. 参见下载包中的“README.md”进行修改、编译和安装。

8.8.2 安装 PyTorch

8.8.2.1 安装前必读

为使用PyTorch框架的开发者提供昇腾AI处理器的超强算力，昇腾开发PyTorch Ascend Adapter插件（`torch_npu`）用于适配PyTorch框架，本章节指导用户在昇腾环境安装PyTorch框架和`torch_npu`插件。

若用户仅进行离线推理，请跳过此章节。

前提条件

- 安装配套版本的CANN软件并配置[CANN环境变量](#)。
- 安装二进制算子包具体请参考[6.5.2 安装二进制算子包](#)。
- 安装对应框架版本的torchvision，PyTorch 1.11.0需安装0.12.0版本，PyTorch 2.0.1版本需安装0.15.2版本，PyTorch 2.1.0版本需安装0.16.0版本。

执行如下命令安装torchvision，以0.12.0版本为例：

```
pip3 install torchvision==0.12.0
```

- 通过源码编译安装torch_npu插件时，安装如下环境依赖。

执行如下命令安装。如果使用非root用户安装，需要在命令后加`--user`，例如：

pip3 install pyyaml --user。

```
pip3 install pyyaml
pip3 install wheel
pip3 install setuptools
```

版本配套关系

表 8-5 PyTorch 版本配套关系

CANN版本	PyTorch版本	PyTorch Ascend Adapter 插件版本	PyTorch Ascend Adapter 插件代码分支名称	PyTorch Ascend Adapter 插件安装包版本	Python版本	AscendHub 镜像版本/名称
7.0.0.alpha003	PyTorch1.11.0	5.0.0	v1.11.0-5.0.0	1.11.0.post8	Python3.7.x（3.7.5及以上）、Python3.8.x、Python3.9.x、Python3.10.x	-
	PyTorch2.0.1		v2.0.1-5.0.0	2.0.1.post1	Python3.8.x、Python3.9.x、Python3.10.x	-
	PyTorch2.1.0		v2.1.0-5.0.0	2.1.0	Python3.8.x、Python3.9.x、Python3.10.x	-

安装步骤

须知

安全提示：建议使用非管理员账户安装运行torch_npu，且建议对安装程序的目录文件做好权限管控，建议文件夹权限设置为750，文件权限设置为 640。可以通过设置umask控制安装后文件的权限，如设置umask为0027。更多安全相关内容请参考[8.8.2.5 安全说明](#)。

在昇腾环境安装PyTorch步骤如下：

1. 安装PyTorch框架。
2. 安装torch_npu插件：可以通过源码编译安装。
3. （可选）安装APEX模块：混合精度训练时安装。

说明

- 如果用户不确定自己的软件包版本，可参考[A.8 查询软件包版本信息](#)进行查询。

8.8.2.2 安装 PyTorch 框架

x86_64 架构

 说明

在x86_64架构下，官方torch包使用MKL加速库。如果需要使用其他blas和lapack加速库（如openblas）来提升性能，请使用源码编译安装方式安装官方torch包，步骤请参考[PyTorch官方链接](#)。

方式一：

使用以下命令可以直接从PyTorch官方预编译包的下载页面中，选择与本地环境匹配的版本包来安装PyTorch，包括匹配操作系统平台（例如Linux）和匹配系统的Python版本号（例如cp37）。例如torch==1.11.0+cpu：指定安装torch版本为1.11.0，并且是CPU版本。

- 安装1.11.0版本
pip3 install torch==1.11.0+cpu -f https://download.pytorch.org/whl/torch_stable.html
- 安装2.0.1版本
pip3 install torch==2.0.1+cpu -f https://download.pytorch.org/whl/torch_stable.html
- 安装2.1.0版本
pip3 install torch==2.1.0+cpu -f https://download.pytorch.org/whl/torch_stable.html

方式二：

步骤1 点击下方PyTorch官方链接下载whl包。PyTorch安装包版本需与Python版本一一对应，例如Python版本为3.8.x，需下载"torch-version+cpu-cp38-cp38-linux_x86_64.whl"PyTorch安装包。更多PyTorch版本可访问<https://download.pytorch.org/whl/torch/>查询。

参数说明：

- torch：表示PyTorch。
- version：PyTorch的版本号，例如1.11.0。
- cpu：说明是CPU版本的PyTorch。
- cp38-cp38：适用于Python 3.8的版本。
- linux_x86_64：适用于linux系统的x86_64架构。
- whl：Python编译好的wheel二进制发布包格式。

表 8-6 x86_64 常用下载链接

PyTorch版本	Python版本	下载链接
1.11.0	Python 3.7 和Python 3.7m	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp37-cp37m-linux_x86_64.whl
	Python 3.8	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp39-cp39-linux_x86_64.whl

PyTorch 版本	Python 版本	下载链接
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-1.11.0%2Bcpu-cp310-cp310-linux_x86_64.whl
2.0.1	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp39-cp39-linux_x86_64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp310-cp310-linux_x86_64.whl
2.1.0	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp38-cp38-linux_x86_64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp39-cp39-linux_x86_64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.1.0%2Bcpu-cp310-cp310-linux_x86_64.whl

步骤2 执行以下命令安装。以安装适配Python 3.8的PyTorch 1.11.0版本为例，其他版本请根据下载的软件包修改命令：

```
pip3 install torch-1.11.0+cpu-cp38-cp38-linux_x86_64.whl
```

如果使用非root用户安装，需要在命令后加--user。

----结束

aarch64

步骤1 获取安装包。更多PyTorch版本可访问<https://download.pytorch.org/whl/torch/>查询。PyTorch安装包版本需与Python版本一一对应，例如Python版本为3.8.x，需下载"torch-version-cp38-cp38-manylinux2014_aarch64.whl"PyTorch安装包。其中：

- torch：表示PyTorch。
- version：PyTorch的版本号，例如1.11.0。
- cp38-cp38：表示这个安装包是为Python 3.8编译的。
- manylinux2014_aarch64：表示支持manylinux2014标准的aarch64架构系统(64位ARM)。
- whl：表明这是一个Python wheel格式的二进制发布包。

表 8-7 aarch64 常用下载链接

PyTorch 版本	Python 版本	下载链接
1.11.0	Python 3.7 和Python 3.7m	https://download.pytorch.org/whl/torch-1.11.0-cp37-cp37m-manylinux2014_aarch64.whl

PyTorch版本	Python版本	下载链接
	Python 3.8	https://download.pytorch.org/whl/torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/torch-1.11.0-cp39-cp39-manylinux2014_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/torch-1.11.0-cp310-cp310-manylinux2014_aarch64.whl
2.0.1	Python 3.8	https://download.pytorch.org/whl/torch-2.0.1-cp38-cp38-manylinux2014_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/torch-2.0.1-cp39-cp39-manylinux2014_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/torch-2.0.1-cp310-cp310-manylinux2014_aarch64.whl
2.1.0	Python 3.8	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp38-cp38-manylinux_2_17_aarch64.whl
	Python 3.9	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.whl
	Python 3.10	https://download.pytorch.org/whl/cpu/torch-2.1.0-cp310-cp310-manylinux_2_17_aarch64.whl

用户可使用上方链接直接下载。或者使用wget命令下载，以下载适配Python 3.8的的PyTorch 1.11.0版本为例，执行如下安装命令（其他版本请根据下载的软件包修改命令）：

```
wget https://download.pytorch.org/whl/torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
```

步骤2 执行如下命令安装。以安装适配Python 3.8的的PyTorch 1.11.0版本为例，执行如下安装命令（其他版本请根据下载的软件包修改命令）：

```
pip3 install torch-1.11.0-cp38-cp38-manylinux2014_aarch64.whl
```

如果使用非root用户安装，需要在命令后加--user。

----结束

8.8.2.3 安装 torch_npu 插件

编译安装适用于进行算子适配开发、CANN版本与PyTorch兼容适配场景下。

以下操作步骤以安装PyTorch 2.0.1版本为例。

步骤1 安装依赖。

选择编译安装方式安装时需要安装系统依赖。目前支持CentOS与Ubuntu操作系统。

- CentOS

```
yum install -y patch libjpeg-turbo-devel dos2unix openblas git
yum install -y gcc==7.3.0 cmake==3.12.0
```
- Ubuntu

```
apt-get install -y patch build-essential libbz2-dev libreadline-dev wget curl llvm libncurses5-dev
libncursesw5-dev xz-utils tk-dev liblzma-dev m4 dos2unix libopenblas-dev git
apt-get install -y gcc==7.3.0 cmake==3.12.0
```

gcc7.3.0版本及以上，cmake3.12.0版本及以上；若需要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。

步骤2 编译生成torch_npu插件的二进制安装包。CANN和PyTorch的配套关系请通过[单击这里查看](#)。

1. 下载对应PyTorch版本分支代码，进入插件根目录，以v6.0.rc1.alpha003-pytorch2.0.1为例，其他版本请下载对应PyTorch版本分支代码。

```
git clone -b v6.0.rc1.alpha003-pytorch2.0.1 https://gitee.com/ascend/pytorch.git
cd pytorch
```

2. （可选）如果不需要切换opplugin（CANN）的版本，请跳过此步骤；如需切换opplugin版本，如切换opplugin的 master分支最新代码，则进行以下操作进行切换。

同步submodule，将opplugin等三方库同步到本地。

```
git submodule init && git submodule update
cd third_party/op-plugin/
# 拉取特定版本的代码并切换到该版本，此处以master为例
git fetch origin master && git checkout master
cd ../../
```

3. 编译生成二进制安装包。

```
# 指定Python版本编包方式，以Python3.8为例，其他Python版本请使用 --python=3.9或--python3.10
bash ci/build.sh --python=3.8
```

步骤3 安装pytorch/dist目录下生成的插件torch_npu包，如果使用非root用户安装，需要在命令后加--user。

```
# 请用户根据实际情况更改命令中的torch_npu包名
pip3 install --upgrade dist/torch_npu-2.0.1.post1-cp38-cp38-linux_aarch64.whl
```

步骤4 验证是否安装成功。

```
python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu(); print(a + a);"
```

显示如下回显证明PyTorch框架与插件安装成功。

```
new kernel registered at /home/p217/pytorch-1.10.1/pytorch
tensor([[ -0.6066,  6.3385,  0.0379,  3.3356],
        [ 2.9243,  3.3134, -1.5465,  0.1916],
        [-2.1807,  0.2008, -1.1431,  2.1523]], device='npu:0')
```

---结束

8.8.2.4 安装 APEX 模块

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芯片的架构特性，用户需要开启混合精度，可以提升模型的性能。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。

混合精度的介绍可参考《PyTorch 训练模型迁移调优指南》中的“模型迁移与训练>自动混合精度（AMP）”章节，推荐用户使用框架自带的AMP功能。

APEX模块的使用介绍可参考《PyTorch 训练模型迁移调优指南》中的“模型套件和第三方库>APEX”章节。

推荐用户通过编译源码包安装APEX模块。

步骤1 安装依赖。

选择编译安装方式安装时需要安装系统依赖。目前支持CentOS与Ubuntu操作系统。

- CentOS

```
yum install -y patch libjpeg-turbo-devel dos2unix openblas git
yum install -y gcc==7.3.0 cmake==3.12.0
```
- Ubuntu

```
apt-get install -y patch build-essential libbz2-dev libreadline-dev wget curl llvm libncurses5-dev
libncursesw5-dev xz-utils tk-dev liblzma-dev m4 dos2unix libopenblas-dev git
apt-get install -y gcc==7.3.0 cmake==3.12.0
```

gcc7.3.0版本及以上，cmake3.12.0版本及以上；若需要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。

步骤2 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。

```
pip install setuptools==65.7.0
```

步骤3 获取昇腾适配的APEX源码。

```
git clone -b 5.0.0 https://gitee.com/ascend/apex.git
```

步骤4 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=3.7 # 支持python3.7、3.8、3.9、3.10
```

说明

请确保NPU版本的PyTorch可以正常使用，否则会影响APEX的编译。

步骤5 执行如下命令安装。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python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-cp37-cp37m-linux_aarch64.whl
```

说明

pip版本要求小于等于24.0，以确保APEX正常安装。可参考如下命令进行指定pip版本的安装：

```
python -m pip install pip==version
```

----结束

8.8.2.5 安全说明

漏洞提示

PyTorch 1.11.0存在CVE-2022-45907漏洞，该漏洞因torch.jit.annotations.parse_type_line方法中不安全使用eval方法，存在可以执行任意代码风险。torch_npu已参考[LINK](#)进行修复。

安全加固

网络加固

PyTorch分布式训练服务需要在设备间进行通信，通信开启的端口默认为全0侦听，为了避免安全风险，建议用户针对此场景进行安全加固。如使用iptables配置防火墙，在运行分布式训练开始前限制外部对分布式训练使用端口的访问，在运行分布式训练结束后清理防火墙规则。

- 防火墙规则设定，可参考如下脚本：

```
#!/bin/bash
set -x

# 要限制的端口号
port={端口号}

# 清除旧规则
iptables -D INPUT -p tcp -j {规则名}
iptables -F {规则名}
iptables -X {规则名}

# 创建新的规则链
iptables -t filter -N {规则名}

# 在多机场景下设定白名单，允许其他节点访问主节点的侦听端口
# 在规则链中添加允许特定IP地址范围的规则
iptables -t filter -A {规则名} -i eno1 -p tcp --dport $port -s {允许外部访问的IP} -j ACCEPT

# 屏蔽外部地址访问分布式训练端口
# 在PORT-LIMIT-RULE规则链中添加拒绝其他IP地址的规则
iptables -t filter -A {规则名} -i {要限制的网卡名} -p tcp --dport $port -j DROP

# 将流量传递给规则链
iptables -I INPUT -p tcp -j {规则名}
```

- 防火墙规则移除，可参考如下脚本：

```
#!/bin/bash
set -x

# 清除规则
iptables -D INPUT -p tcp -j {规则名}
iptables -F {规则名}
iptables -X {规则名}
```

防火墙规则设定和移除示例：

1. 针对特定端口设定防火墙，脚本中端口号为要限制的端口，在PyTorch分布式训练中端口号请参考[表3 通信矩阵](#)；要限制的网卡名为服务器用于分布式通信使用的网卡，允许的外部访问的IP为分布式训练服务器的IP地址。网卡和服务器IP可以通过ifconfig查看，如下文回显的eth0为网卡名，192.168.1.1为服务器IP地址：

```
# ifconfig
eth0
    inet addr:192.168.1.1 Bcast:192.168.1.255 Mask:255.255.255.0
    inet6 addr: fe80::230:64ee:ef1a:c1a/64 Scope:Link
```

2. 假定服务器主节点地址192.168.1.1，另一台需要进行分布式训练的服务器为192.168.1.2，训练端口为29510。

- 规则设定可参考如下脚本：

```
#!/bin/bash
set -x

# 设定侦听的端口
port=29510

# 清除旧规则
iptables -D INPUT -p tcp -j PORT-LIMIT-RULE
iptables -F PORT-LIMIT-RULE
iptables -X PORT-LIMIT-RULE

# 创建新的PORT-LIMIT-RULE规则链
iptables -t filter -N PORT-LIMIT-RULE

# 在多机场景下设定白名单，允许192.168.1.2访问主节点
# 在PORT-LIMIT-RULE规则链中添加允许特定IP地址范围的规则
iptables -t filter -A PORT-LIMIT-RULE -i eno1 -p tcp --dport $port -s 192.168.1.2 -j ACCEPT

# 屏蔽外部地址访问分布式训练端口
# 在PORT-LIMIT-RULE规则链中添加拒绝其他IP地址的规则
```

```
iptables -t filter -A PORT-LIMIT-RULE -i eth0 -p tcp --dport $port -j DROP

# 将流量传递给PORT-LIMIT-RULE规则链
iptables -I INPUT -p tcp -j PORT-LIMIT-RULE

- 规则移除可参考如下脚本：
#!/bin/bash
set -x
# 清除规则
iptables -D INPUT -p tcp -j PORT-LIMIT-RULE
iptables -F PORT-LIMIT-RULE
iptables -X PORT-LIMIT-RULE
```

权限控制

- 建议用户对训练所需文件、训练过程中保存的文件、用户个人的隐私数据、商业资产等敏感文件做好权限控制等安全措施，例如多用户共享数据集场景下的数据集文件写权限控制、profiler等场景产生数据文件权限控制等，设定的权限建议参考[表1 文件权限参考](#)进行设置。
- torch_npu中profiler工具会生成性能记录文件，生成的文件权限为640，文件夹权限为750，用户可根据需要自行对生成后的相关文件进行权限控制。
- 用户安装和使用过程需要做好权限控制，建议参考[表1 文件权限参考](#)进行设置。如需要保存安装/卸载日志，可在安装/卸载命令后面加上参数--log <FILE>，注意对<FILE>文件及目录做好权限管控。
- PyTorch框架运行中所生成的文件权限依赖系统设定，如torch.save接口保存的文件。建议当前执行脚本的用户根据自身需要，对生成文件做好权限控制，设定的权限可参考[表1 文件权限参考](#)。可使用umask控制默认权限，避免提权等安全风险，建议用户将主机（包括宿主机）和容器中的umask设置为027及其以上，提高安全性。

文件权限参考

建议用户根据自身需要，参考[表1 文件权限参考](#)对各类文件进行加固。

表 8-8 文件权限参考

类型	linux权限参考值	备注
文件夹 / 目录	750 (rwxr-x---)	对于共享目录可为755
数据集文件	640 (rw-r-----)	对于共享数据集文件可为644
checkpoint文件	640 (rw-r-----)	对于checkplint等生成文件可以设置为640
程序文件	440 (r--r-----)	除非开发调试场景，正常运行时程序文件不应再次修改
可执行脚本	750 (rwxr-x---)	针对可执行脚本设置750

网络通信

PyTorch提供分布式训练能力，支持在单机和多机场景下进行训练，需要进行网络通信。其中PyTorch需要使用TCP进行通信，torch_npu使用CANN中HCCL在NPU设备间通信，通信端口见[表8-10](#)。用户需要注意并保障节点间通信网络安全，可以使用iptables等方式消减安全风险，可参考[安全加固](#)章节进行网络安全加固。

数据安全

- PyTorch和torch_npu提供的profiler性能分析工具和torch_npu提供AOE性能调优工具都会在本地生成性能拆解数据。用户需加强对相关数据的保护，需要模型性能调优时使用，调优完成后及时关闭，AOE和Profiler工具具体细节请参考《PyTorch 训练模型迁移调优指南》。
- PyTorch使用过程中需要加载和保存数据，部分接口使用风险模块pickle，可能存在数据风险，如torch.load、torch.distributed.scatter_object_list等接口，可参考[torch.load](#)、[collective-functions](#)了解具体风险。
- AscendPyTorch依赖CANN的基础能力实现AOE性能调优、算子dump、日志记录等功能，用户需关注上述功能生成文件的权限控制。

编译安全

torch_npu支持源码编译安装，在编译时会下载依赖第三方库并执行构建shell脚本，在编译过程中会产生临时程序文件和编译目录。用户可根据需要自行对源代码目录内的文件进行权限管控避免安全风险。

运行提示

- 建议根据运行环境资源状况编写训练脚本。若训练脚本与资源状况不匹配，如数据集的size超出内存容量/NPU存储容量等，可能引发错误并导致进程退出。
- PyTorch和torch_npu在运行异常时会退出进程并打印报错信息，建议根据报错提示定位具体错误原因，包括设定算子同步执行、查看CANN日志、解析生成的Core Dump文件等方式。

公网地址

在torch_npu的配置文件和脚本中存在[表8-9](#)：

表 8-9 公网地址

类型	开源代码地址	文件名	公网IP地址/公网URL地址/域名/邮箱地址	用途说明
自研	不涉及	ci\docker\X86\Dock erfile	https://mirrors.huaweicloud.com/repository/pypi/simple	docker配置文件，用于配置pip源
自研	不涉及	ci\docker\X86\Dock erfile	https://download.pytorch.org/whl/cpu	docker配置源，用于配置torch下载连接
自研	不涉及	ci\docker\ARM\ Dockerfile	https://mirrors.aliyun.com/pypi/simple	docker配置文件，用于配置pip源
自研	不涉及	setup.cfg	https://gitee.com/ascend/pytorch	用于打包whl的url入参

类型	开源代码地址	文件名	公网IP地址/公网URL地址/域名/邮箱地址	用途说明
自研	不涉及	setup.cfg	https://gitee.com/ascend/pytorch/tags	用于打包whl的download_url入参
自研	不涉及	third_party\op-plugin\ci\build.sh	https://gitee.com/ascend/pytorch.git	编译脚本根据torch_npu仓库地址拉取代码进行编译
自研	不涉及	third_party\op-plugin\ci\exec_ut.sh	https://gitee.com/ascend/pytorch.git	UT脚本根据torch_npu仓库地址下拉取代码进行UT测试
开源引入	https://github.com/pytorch/pytorch/blob/main/test/nn/test_convolution.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_mps.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/legacy_conv2d.pt	用于test脚本下载相关pt文件
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/legacy_serialized.pt	用于test脚本下载相关pt文件
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/gpu_tensors.pt	用于test脚本下载相关pt文件

类型	开源代码地址	文件名	公网IP地址/公网URL地址/域名/邮箱地址	用途说明
开源引入	https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_nn.py https://github.com/pytorch/pytorch/blob/v2.1.0/test/test_serialization.py	test\url.ini	https://download.pytorch.org/test_data/linear.pt	用于test脚本下载相关pt文件

接口说明

AscendPyTorch是PyTorch适配插件，支持用户使用PyTorch在昇腾设备上进行训练和推理。AscendPyTorch适配后支持用户使用PyTorch原生接口。除了原生PyTorch接口外，AscendPyTorch提供了部分自定义接口，包括自定义算子、亲和库和其他接口，支持PyTorch接口和自定义接口连接，具体可参考《PyTorch 训练模型迁移调优指南》>API列表。

参考[PyTorch社区公开接口规范](#)，PyTorchAdapter提供了对外的自定义接口。如果一个函数看起来符合公开接口的标准且在文档中有展示，则该接口是公开接口。否则，使用该功能前可以在社区询问该功能是否确实是公开的或意外暴露的接口，因为这些未暴露接口将来可能会被修改或者删除。

AscendPyTorch使用monkey-patch方式实现PyTorch接口的适配，通过将torch_npu中的部分接口动态替换PyTorch中的接口实现对部分接口的支持。推荐用户使用《PyTorch 训练模型迁移调优指南》>API列表中推荐的调用方式。

AscendPyTorch项目采用C++和Python联合开发，当前除Libtorch场景外正式接口只提供Python接口，在torch_npu的二进制包中动态库不直接提供服务，暴露的接口为内部使用，不建议用户使用。

通信矩阵

通信矩阵具体可参考下表。

表 8-10 通信矩阵

组件	PyTorch	HCCL
源设备	运行torch_npu进程的服务器	运行torch_npu进程的服务器
源IP	设备地址IP	设备地址IP

源端口	操作系统自动分配，分配范围由操作系统的自身配置决定	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
目的设备	运行torch_npu进程的服务器	运行torch_npu进程的服务器
目的IP	设备地址IP	设备地址IP
目的端口（侦听）	默认29500/29400，用户可以设定端口号	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
协议	tcp	tcp
端口说明	- 在静态分布式场景中（用torchrun/torch.distributed.launch使用的backend为static），目的端口（默认29500）用于接收和发送数据，源端口用于接收和发送数据 - 在动态分布式场景中（用torchrun或者torch.distributed.launch使用的backend为c10d），目的端口（默认29400）用于接收和发送数据，源端口用于接收和发送数据。可以使用rdzv_endpoint和master_port指定端口号 - 在分布式场景中，用torchrun或者torch.distributed.launch不指定任何参数时使用的端口号为29500	默认值为60000，取值范围[0,65520]。用户可以通过HCCL_IF_BASE_PORT环境变量指定Host网卡起始端口号，配置后系统默认占用以该端口起始的16个端口
侦听端口是否可更改	是	是
认证方式	无认证方式	无认证方式
加密方式	无	无

所属平面	不涉及	不涉及
版本	所有版本	所有版本
特殊场景	无	无
备注	该通信过程由开源软件 PyTorch 控制，配置为 PyTorch 原生设置，可参考 PyTorch 文档 。源端口由操作系统自动分配，分配范围由操作系统的配置决定，例如ubuntu：采用 /proc/sys/net/ipv4/ipv4_local_port_range 文件指定，可通过 cat /proc/sys/net/ipv4/ipv4_local_port_range 或 sysctl net.ipv4.ip_local_port_range 查看	该通信过程由CANN中 HCCL 组件控制，torch_npu 不进行控制，端口范围可参考《环境变量参考》的“执行相关 > 集合通信与分布式训练 > 集合通信相关配置 > HCCL_IF_BASE_PORT”和《集合通信接口参考》

8.8.3 安装昇思 MindSpore

若用户仅进行离线推理，请跳过此章节。

用户要使用 MindSpore 框架，请登录 [MindSpore 官网](#) 获取安装 MindSpore 框架的方法。

8.9 安装 Python 版本的 proto

如果训练脚本依赖 protobuf 的 Python 版本进行序列化结构的数据存储（例如 TensorFlow 的序列化相关接口），则需要安装 Python 版本的 proto。

步骤1 检查系统中是否存在“/usr/local/python3.7.5/lib/python3.7/site-packages/google/protobuf/pyext/_message.cpython-37m-*<arch>*-linux-gnu.so”（以参考 [8.4 安装依赖](#) 章节安装的 python3.7.5 为例）这个动态库，如果没有，需要按照如下步骤安装。其中 *<arch>* 为系统架构类型。

说明

“/usr/local/python3.7.5/lib/python3.7/site-packages” 是 pip 安装第三方库的路径，可以使用 **pip3 -V** 检查。

如果系统显示：/usr/local/python3.7.5/lib/python3.7/site-packages/pip，则 pip 安装第三方库的路径为 /usr/local/python3.7.5/lib/python3.7/site-packages。

步骤2 执行如下命令卸载 protobuf。

```
pip3 uninstall protobuf
```

步骤3 下载 protobuf 软件包。

从 <https://github.com/protocolbuffers/protobuf/releases/download/v3.11.3/protobuf-python-3.11.3.tar.gz> 路径下下载 3.11.3 版本 protobuf-python-3.11.3.tar.gz

软件包（或者其他版本，保证和当前环境上安装的tensorflow兼容），并以root用户上传到所在Linux服务器任意目录并执行命令解压。

```
tar zxvf protobuf-python-3.11.3.tar.gz
```

步骤4 以root用户安装protobuf。

进入protobuf软件包目录。

1. 安装protobuf的依赖。

当操作系统为Ubuntu时，安装命令如下：

```
apt-get install autoconf automake libtool curl make g++ unzip libffi-dev -y
```

当操作系统为CentOS/BClinux时，安装命令如下：

```
yum install autoconf automake libtool curl make gcc-c++ unzip libffi-devel -y
```

2. 为autogen.sh脚本添加可执行权限并执行此脚本。

```
chmod +x autogen.sh
```

```
./autogen.sh
```

3. 配置安装路径（默认安装路径为"/usr/local"）。

```
./configure
```

如果想指定安装路径，可参考以下命令。

```
./configure --prefix=/protobuf
```

“/protobuf”为用户指定的安装路径。

4. 执行protobuf安装命令。

```
make -j15 # 通过grep -w processor /proc/cpuinfo|wc -l查看cpu数，示例为15，用户可自行设置相应参数。
```

```
make install
```

5. 刷新共享库。

```
ldconfig
```

protobuf安装完成后，会在[步骤4.3](#)中配置的路径下面的include目录中生成google/protobuf文件夹，存放protobuf相关头文件；在[步骤4.3](#)中配置路径下面的bin目录中生成protoc可执行文件，用于进行*.proto文件的编译，生成protobuf的C++头文件及实现文件。

6. 检查是否安装完成。

```
ln -s /protobuf/bin/protoc /usr/bin/protoc
```

```
protoc --version
```

其中/protobuf为[步骤4.3](#)中用户配置的安装路径。如果用户未配置安装路径，则直接执行protoc --version检查是否安装成功。

步骤5 安装protobuf的python版本运行库。

1. 进入protobuf软件包目录的python子目录，编译python版本的运行库。

```
python3 setup.py build --cpp_implementation
```

📖 说明

这里需要编译二进制版本的运行库，如果使用python3 setup.py build命令无法生成二进制版本的运行库，在序列化结构的处理时性能会非常慢。

2. 安装动态库。

```
cd .. && make install
```

进入python子目录，安装python版本的运行库。

```
python3 setup.py install --cpp_implementation
```

3. 检查是否安装成功。

检查系统中是否存在“/usr/local/python3.7.5/lib/python3.7/site-packages/protobuf-3.11.3-py3.7-linux-aarch64.egg/google/protobuf/pyext/_message.cpython-37m-<arch>-linux-gnu.so”这个动态库。其中<arch>为系统架构类型。

说明

“/usr/local/python3.7.5/lib/python3.7/site-packages”是pip安装第三方库的路径，可以使用**pip3 -V**检查。

如果系统显示：/usr/local/python3.7.5/lib/python3.7/site-packages/pip，则pip安装第三方库的路径为/usr/local/python3.7.5/lib/python3.7/site-packages。

4. 若用户在**步骤4.3**中指定了安装路径，则需在运行脚本中增加环境变量的设置：

```
export LD_LIBRARY_PATH=/protobuf/lib:${LD_LIBRARY_PATH}
```

“/protobuf”为**步骤4.3**中用户配置的安装路径。

5. 建立软连接。

当用户自行配置安装路径时，需要建立软连接，否则导入tensorflow会报错。命令如下：

```
ln -s /protobuf/lib/libprotobuf.so.22.0.3 /usr/lib/libprotobuf.so.22
```

其中“/protobuf”为**步骤4.3**中用户配置的安装路径。

----结束

8.10 配置 device 的网卡 IP

当进行分布式训练时，需要通过昇腾软件中的HCCN Tool工具配置device的网卡IP，用于多个device间通信以实现网络模型参数的同步更新。如果用户需要使用HCCN Tool工具的其他功能（如检查网口Link状态），请参见《[Atlas 中心训练服务器 23.0.x\(23.0.3及其系列版本\) HCCN Tool 接口参考](#)》或《[Atlas A2 中心推理和训练硬件 23.0.x\(23.0.3及其系列版本\) HCCN Tool 接口参考](#)》。

Atlas 800 训练服务器、Atlas 900 AI 集群、Atlas 300T 训练卡

Atlas 800 训练服务器（型号 9000）、Atlas 800 训练服务器（型号 9010）、Atlas 900 AI集群（型号 9000）、Atlas 300T 训练卡（型号 9000）、Atlas 300T Pro 训练卡（型号：9000）请参考《[Ascend Training Solution 23.0.RC1 组网指南](#)》的“参数面网络介绍>中大规模集群组网/小规模组网>参数面网络配置示例”章节中“配置训练节点”，通过HCCN_Tool配置NPU网口的device IP。

Atlas A2 训练系列产品

Atlas 800T A2 训练服务器、Atlas 900 A2 PoD 集群基础单元A900 A2 PoD 集群基础单元、Atlas 200T A2 Box16 异构子框请参考《[Ascend Training Solution 23.0.0 组网指南](#)》的“参数面网络配置介绍>配置示例（选择对应的产品）>配置训练节点”，通过HCCN_Tool配置NPU网口的device IP。

9 升级

- 9.1 升级前必读
- 9.2 升级操作

9.1 升级前必读

升级影响

- 升级过程禁止进行其他维护操作动作。
- 软件版本升级过程中会导致业务中断。
- 升级软件包后，不会影响正常业务。

注意事项

软件版本升级时的注意事项如[表9-1](#)所示。

表 9-1 升级时注意事项

序号	描述
1	在进行升级操作之前，请仔细阅读本文档，确定已经理解全部内容。如果您对文档有任何意见或建议，请联系华为技术支持解决。
2	为了减少对业务的影响，请提前切走业务或在业务量低时进行升级操作。
3	升级后，请确保所有软件的版本保持一致。
4	支持多版本情况下升级，但默认只升级安装目录下latest软链接指向的版本（即当前版本）。
5	如果需从高版本回退至低版本，需卸载高版本后重新安装所需版本。

 说明

运行环境如果安装的是nnrt软件包，如果需要支持在线推理，请先卸载nnrt，再安装nnae。

版本要求

驱动版本、固件版本需与CANN软件版本保持配套关系。版本配套关系请参见[这里](#)。

9.2 升级操作

升级流程

- 在驱动和固件和CANN软件配套场景下，可单独升级CANN软件。
- 如果也要升级固件和驱动，请按照“固件->驱动->CANN软件”的顺序进行升级。

升级驱动和固件

请根据[表9-2](#)参考对应文档进行升级操作。

表 9-2 驱动和固件升级指导

产品型号	参考文档
Atlas 200（EP场景）	请参见《 Atlas 200 AI加速模块 6.0.0 NPU驱动和固件升级指南（EP场景） 》。
Atlas 200（RC场景）	请参见《 Atlas 200 AI加速模块 6.0.0 软件安装与维护指南（RC场景） 》。
Atlas 200I A2（EP场景）	请参见《 NPU驱动和固件升级指南（EP场景） 》。
Atlas 200I A2（RC场景）	请参见《 Atlas 200I A2 加速模块 23.0.x(23.0.3及其系列版本) 驱动开发指南 》。
Atlas 500 A2	请参见《 Atlas 500 A2 智能小站 用户指南 》。
Atlas 200I SoC A1	请参见《 Atlas 200I SoC A1核心板 23.0.3 NPU驱动和固件升级指南 》。
A300-3000	请参见《 Atlas 300I 推理卡 23.0.3 NPU驱动和固件升级指南（型号 3000, 3010） 》。
A300-3010	请参见《 Atlas 300I 推理卡 23.0.3 NPU驱动和固件升级指南（型号 3000, 3010） 》。
Atlas 300I Pro	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件升级指南 》。
Atlas 300V Pro	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件升级指南 》。
Atlas 300V	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件升级指南 》。

产品型号	参考文档
Atlas 300I Duo	请参见《 Atlas 中心推理卡 23.0.3 NPU驱动和固件升级指南 》。
Atlas 200T A2 Box16	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件升级指南 》。
Atlas 300T A2	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件升级指南 》。
A300T-9000	请参见《 Atlas 中心训练卡 23.0.3 NPU驱动和固件安装指南 》。
Atlas 300T Pro	请参见《 Atlas 中心训练卡 23.0.3 NPU驱动和固件安装指南 》。
A500 Pro-3000	请参见所配置标卡对应的文档。
A800-3000	请参见所配置标卡对应的文档安装驱动和固件。
A800-3010	请参见所配置标卡对应的文档。
A800-9000	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件升级指南 》。
A800-9010	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件升级指南 》。
A900-9000	请参见《 Atlas 中心训练服务器 23.0.3 NPU驱动和固件升级指南 》。
Atlas 800T A2	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件升级指南 》。
Atlas 900 A2 PoD	请参见《 Atlas A2 中心推理和训练硬件 23.0.3 NPU驱动和固件升级指南 》。

升级 CANN 软件

 说明

升级开发套件包前请确保安装目录可用空间大于9G，如不满足请清理空间或更换安装目录。

CANN软件升级操作如下：

- 步骤1 以软件包的安装用户登录软件包的安装环境。
- 步骤2 进入软件包所在路径。
- 步骤3 增加对软件包的可执行权限。
`chmod +x 软件包名.run`
- 步骤4 执行如下命令校验软件包安装文件的一致性和完整性。
`./软件包名.run --check`
- 步骤5 软件包升级。

- 从CANN 3.2.1升级到当前版本。
 - 若用户安装时指定了安装路径，执行命令如下：

```
./软件包名.run --upgrade --install-path=<path>
```

其中<path>为用户指定的软件包安装目录，软件包名请根据实际包名进行替换。
 - 若用户安装时未指定安装路径，执行命令如下：

```
./软件包名.run --upgrade
```
- 从CANN 3.3.0起低版本升级到高版本，可自动从安装配置文件中导入原有配置信息，无需输入任何安装参数。执行命令如下：

```
./软件包名.run --upgrade
```

用户想使用默认签署[华为企业业务最终用户许可协议（EULA）](#)的方式升级软件包时，可以添加--quiet参数配合升级命令使用，如：

```
./软件包名.run --upgrade --quiet
```

添加该参数后会跳过[步骤6](#)的确认操作。

步骤6 用户需签署[华为企业业务最终用户许可协议（EULA）](#)后进入升级流程，根据回显页面执行y或Y确认协议，输入其他任意字符为拒绝协议，确认接受协议后开始升级。

若当前语言环境不满足要求，可以执行如下命令配置系统的默认语言环境。

```
#配置为中文（简体）
export LANG=zh_CN.UTF-8
#配置为英文
export LANG=en_US.UTF-8
```

----结束

10 卸载

如果用户只是卸载CANN软件包（如nnrt、toolkit等），那卸载没有先后顺序，但是如果也要卸载驱动和固件，则需要卸载其他软件包以后再卸载驱动和固件。

通过*.run格式安装CANN软件后，可参考以下两种方式进行卸载。

说明

安装时，如果以修改~/.bashrc文件方式设置了永久环境变量（如：`./usr/local/Ascend/ascend-toolkit/set_env.sh`），卸载完成后建议在~/.bashrc文件删除对应指令。

方法一 脚本卸载

用户可以通过卸载脚本完成卸载。

步骤1 进入软件的卸载脚本所在路径，一般放置在“script”目录下（“script”所在路径请以实际为准）。

以toolkit和nnae软件包为例：

- 进入toolkit软件的卸载脚本所在路径（nnrt目录结构同toolkit）
`cd <path>/ascend-toolkit/<version>/<arch>-linux/script`
- 进入nnae软件的卸载脚本所在路径（tfplugin目录结构同nnae）
`cd <path>/nnae/<version>/script`

其中<path>为软件包的安装路径，<version>为软件包版本，<arch>-linux为CPU架构，请用户根据实际情况替换。

步骤2 执行./uninstall.sh命令运行脚本，完成卸载。

卸载完成后，若显示如下信息，则说明软件卸载成功：

```
[INFO] xxx uninstall success
```

xxx表示卸载的实际软件包名。

----结束

方法二 软件包卸载

如果用户想要对已安装的软件包进行卸载，可以执行如下步骤：

步骤1 以软件包的安装用户登录软件包的安装环境。

步骤2 进入软件包所在路径。

步骤3 执行以下命令卸载软件包。

```
./软件包名.run --uninstall
```

卸载完成后，若显示如下信息，则说明软件卸载成功：

```
[INFO] xxx uninstall success
```

xxx表示卸载的实际软件包名。

----结束

11 后续任务

- 可从[ModelZoo](#)获取离线模型或训练脚本。
- 如需进行推理应用开发，可参见《CANN 应用软件开发指南 (C&C++)》或《[CANN 应用软件开发指南 \(Python\)](#)》在开发环境上开发应用程序。
- 如需进行模型迁移，可参考《CANN TensorFlow网络模型迁移&训练指南》或《PyTorch网络模型移植&训练指南》。

12 安全加固

12.1 加固须知

12.2 Host侧加固

12.1 加固须知

本文中列出的安全加固措施为基本的加固建议项。用户应根据自身业务，重新审视整个系统的网络安全加固措施。用户应按照所在组织的安全策略进行相关配置，包括并不局限于软件版本、口令复杂度要求、安全配置（协议、加密套件、密钥长度等），权限配置、防火墙设置等。必要时可参考业界优秀加固方案和安全专家的建议。

12.2 Host 侧加固

12.2.1 禁止使用 root 账户远程登录系统

root是Linux系统中的超级特权用户，具有所有Linux系统资源的访问权限。如果允许直接使用root账号登录Linux系统对系统进行操作，会带来很多潜在的安全风险。通常情况下，建议将在“/etc/ssh/sshd_config”文件中将“PermitRootLogin”参数设置为“no”，设置后root用户无法通过SSH登录到系统，增加了系统的安全性。如果需要使用root权限进行管理操作，可以通过其他普通用户登录系统后，再使用su或sudo命令切换到root用户进行操作。这样可以避免直接使用root用户登录系统，从而减少系统被攻击的风险。

12.2.2 账户 UID 安全加固

UID为0的账户具有系统最高权限，应保证只有一个，其中只有root是Linux系统中的超级特权用户，确保当前系统中只有root账户uid=0，同时，各系统账号的UID必须是不同的，可以执行id查询用户UID。

12.2.3 内存地址随机化机制安全加固

ASLR（address space layout randomization）开启后能增强漏洞攻击防护能力，建议用户将/proc/sys/kernel/randomize_va_space里面的值设置为2，开启该功能。

12.2.4 禁止使用 SetUID 或 SetGID 的 shell 脚本

特殊权限的脚本有可能被恶意使用，给系统带来巨大的威胁，如非必要，建议禁止使用SUID、SGID脚本。

可以执行如下命令查找系统中SUID/SGID文件，查看是否必要存在。若无必要，去掉“s”位以取消文件的SetUID或SetGID权限，或删除文件。

```
find / -perm -2000 -exec ls -l {} \; -exec md5sum {} \;  
find / -perm -4000 -exec ls -l {} \; -exec md5sum {} \;
```

12.2.5 无属主文件安全加固

用户可以执行**find / -nouser -o -nogroup**命令，查找系统中的无属主文件。根据文件的uid和gid创建相应的用户和用户组，或者修改已有用户的uid、用户组的gid来适配，赋予文件属主，避免无属主文件给系统带来安全隐患。

12.2.6 设置 umask

建议用户将主机（包括宿主机）和容器中的umask设置为0027及其以上，提高安全性。

以设置umask为0077为例，具体操作如下所示。

步骤1 以root用户登录服务器，编辑“/etc/profile”文件。

```
vim /etc/profile
```

步骤2 在“/etc/profile”文件末尾加上**umask 0077**，保存并退出。

步骤3 执行如下命令使配置生效。

```
source /etc/profile
```

----结束

A 常用操作

A.1 安装、回退和卸载 CANN 补丁包

CANN软件包支持补丁升级，即支持独立升级单个或多个库文件的功能。因此发布CANN补丁包。

约束

- 补丁仅能支持对应的基线版本或相关的补丁版本进行升级。
- 基于同一基线版本的补丁，需保证后续安装的补丁版本大于之前安装的补丁版本。
- 仅支持回退一个补丁版本。

安装、回退影响

- 安装、回退过程禁止进行其他维护操作动作。
- 补丁包安装、回退过程中会导致业务中断。
- 补丁包安装、回退后，不会影响正常业务。

CANN 补丁包相关操作（适用于.run 格式）

CANN补丁包支持安装、回退等操作，用户根据实际需要选择相应参数。参数说明请参见[表A-1](#)。

以安装冷补丁为例，命令参考如下（请注意将命令中的**软件包名.run**替换为实际包名）：

```
./软件包名.run --install
```

如需指定补丁包安装路径，可添加“--install-path”参数指定安装路径（如“/usr/local/Ascend”）。

表 A-1 补丁包支持参数说明

参数	说明
--help -h	查询帮助信息。
--version	查询软件包版本。
--info	查询软件包构建信息。
--list	查询软件包文件列表。
--check	检查软件包的一致性和完整性。
--quiet	静默安装，跳过交互式信息。
--nox11	安装过程中不弹出图形终端窗口。
--noexec	解压软件包到当前目录，但不执行安装脚本。配套--extract=<path>使用，格式为： --noexec --extract=<path>
--extract=<path>	解压软件包中文件到指定目录。
--tar arg1 [arg2 ...]	对软件包执行tar命令，使用tar后面的参数作为命令的参数。例如执行--tar xvf命令，解压run安装包的内容到当前目录。
--install	安装补丁包。
--rollback	回退到前一版本。
--uninstall	清除原版本的备份库文件，不支持回退。
--install-path=<path>	指定补丁包安装路径。 若通过该参数指定了安装路径，运行用户需要对指定的安装路径有可读写权限。 若不指定： • 优先读取全局配置文件“ascend_cann_install.info”中的安装路径。 可在如下目录查看是否存在该文件： - root用户：/etc/Ascend - 非root用户：\${HOME}/Ascend • 如果当前环境不存在“ascend_cann_install.info”，则将安装到默认路径下： - 若使用root用户安装，默认安装路径为：/usr/local/Ascend - 若使用非root用户安装，默认安装路径为：\${HOME}/Ascend

A.2 编译安装 Python3.7.5

以安装Python3.7.5为例演示编译安装操作。

步骤1 使用wget下载python3.7.5源码包，可以下载到安装环境的任意目录，命令为：

```
wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
```

步骤2 进入下载后的目录，解压源码包，命令为：

```
tar -zxvf Python-3.7.5.tgz
```

步骤3 进入解压后的文件夹，执行配置、编译和安装命令：

```
cd Python-3.7.5
./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
make
sudo make install
```

其中“--prefix”参数用于指定python安装路径，用户根据实际情况进行修改。“--enable-shared”参数用于编译出libpython3.7m.so.1.0动态库。“--enable-loadable-sqlite-extensions”参数用于加载libssqlite3-dev依赖。

以--prefix=/usr/local/python3.7.5路径为例进行说明。执行配置、编译和安装命令后，安装包在/usr/local/python3.7.5路径，libpython3.7m.so.1.0动态库在/usr/local/python3.7.5/lib/libpython3.7m.so.1.0路径。

步骤4 设置python3.7.5环境变量。

```
#用于设置python3.7.5库文件路径
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
#如果用户环境存在多个python3版本，则指定使用python3.7.5版本
export PATH=/usr/local/python3.7.5/bin:$PATH
```

通过以上export方式设置环境变量，该种方式设置的环境变量只在当前窗口有效。您也可以通过将以上命令写入~/.bashrc文件中，然后执行**source ~/.bashrc**命令，使上述环境变量永久生效。注意如果后续您有使用环境上其他python版本的需求，则不建议将以上命令写入到~/.bashrc文件中。

步骤5 安装完成之后，执行如下命令查看安装版本，如果返回相关版本信息，则说明安装成功。

```
python3 --version
pip3 --version
```

----结束

A.3 安装 3.5.2 版本 cmake

步骤1 使用wget下载cmake源码包，可以下载到安装服务器任意目录，命令为：

```
wget https://cmake.org/files/v3.5/cmake-3.5.2.tar.gz --no-check-certificate
```

步骤2 进入下载后的目录，解压源码包，命令为：

```
tar -zxvf cmake-3.5.2.tar.gz
```

步骤3 进入解压后的文件夹，执行配置，编译和安装命令：

```
cd cmake-3.5.2
./bootstrap --prefix=/usr
make
sudo make install
```

步骤4 安装完成后重新执行**cmake --version**查看版本号。

----结束

A.4 安装 7.3.0 版本 gcc

以下步骤请在root用户下执行。

步骤1 下载gcc-7.3.0.tar.gz。

步骤2 安装gcc时候会占用大量临时空间，所以先执行下面的命令清空/tmp目录：

```
rm -rf /tmp/*
```

步骤3 安装依赖（以CentOS和Ubuntu系统为例）。

- CentOS执行如下命令安装。
yum install bzip2
- Ubuntu执行如下命令安装。
apt-get install bzip2

步骤4 编译安装gcc。

1. 进入gcc-7.3.0.tar.gz源码包所在目录，解压源码包，命令为：

```
tar -zxvf gcc-7.3.0.tar.gz
```

2. 进入解压后的文件夹，执行如下命令下载gcc依赖包：

```
cd gcc-7.3.0  
./contrib/download_prerequisites
```

如果执行上述命令报错，需要执行如下命令在“gcc-7.3.0/”文件夹下下载依赖包：

```
wget http://gcc.gnu.org/pub/gcc/infrastructure/gmp-6.1.0.tar.bz2  
wget http://gcc.gnu.org/pub/gcc/infrastructure/mpfr-3.1.4.tar.bz2  
wget http://gcc.gnu.org/pub/gcc/infrastructure/mpc-1.0.3.tar.gz  
wget http://gcc.gnu.org/pub/gcc/infrastructure/isl-0.16.1.tar.bz2
```

下载好上述依赖包后，重新执行以下命令：

```
./contrib/download_prerequisites
```

如果上述命令校验失败，需要确保依赖包为一次性下载成功，无重复下载现象。

3. 执行配置、编译和安装命令：

```
./configure --enable-languages=c,c++ --disable-multilib --with-system-zlib --prefix=/usr/local/gcc7.3.0  
make -j15 # 通过grep -w processor /proc/cpuinfo|wc -l查看cpu数，示例为15，用户可自行设置相应参数。  
make install
```

说明

其中“--prefix”参数用于指定gcc7.3.0安装路径，用户可自行配置，但注意不要配置为“/usr/local”及“/usr”，因为会与系统使用软件源默认安装的gcc相冲突，导致系统原始gcc编译环境被破坏。示例指定为“/usr/local/gcc7.3.0”。

步骤5 配置环境变量（请在实际需要时再进行配置）。

例如用户在启动在线推理或训练进程前需执行如下命令配置环境变量。

```
export LD_LIBRARY_PATH=/usr/local/gcc7.3.0/lib64:${LD_LIBRARY_PATH}
```

其中“/usr/local/gcc7.3.0”为**步骤4.3**中配置的gcc7.3.0安装路径，请根据实际情况替换。

----结束

A.5 配置网卡 IP 地址

安装完操作系统后，需在当前iBMC远程管理界面中配置网卡IP地址才能远程连接服务器，配置方法如下（以CentOS、Ubuntu系统为例）。

CentOS/EulerOS 操作系统配置网卡 IP 地址

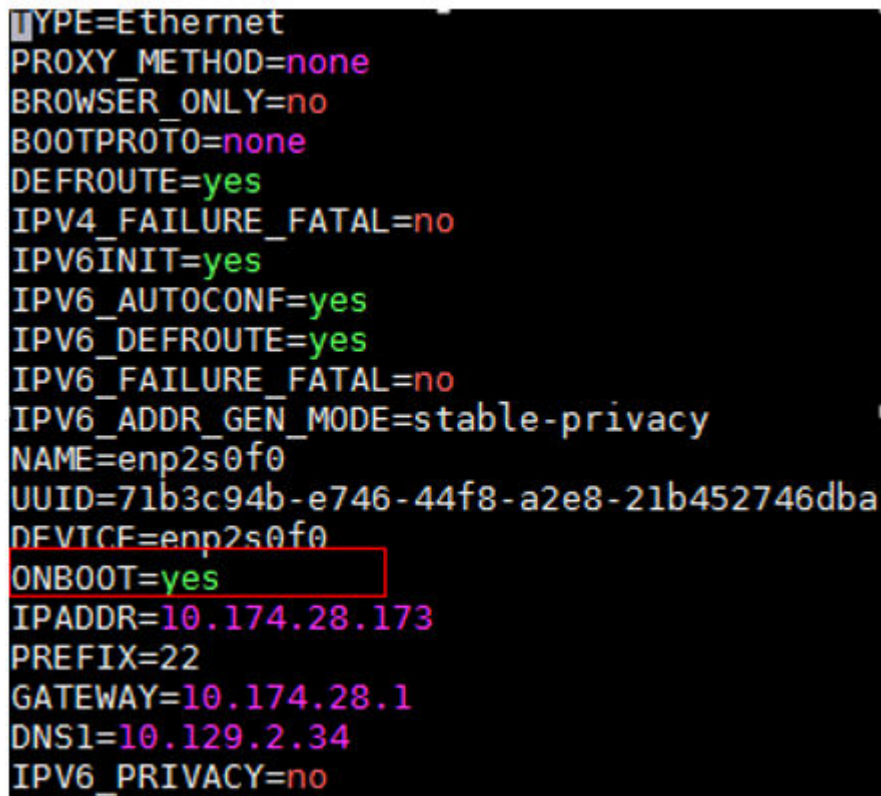
步骤1 以root用户进入OS界面。

步骤2 进入需配置IP地址的网卡的配置文件，此处以网卡名为enp2s0f0举例，具体网卡和路径请以实际环境为准。

```
vi /etc/sysconfig/network-scripts/ifcfg-enp2s0f0
```

步骤3 配置对应网卡的业务IP地址、子网掩码和网关，具体以实际为准，如图A-1所示。

图 A-1 设置网络参数



```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp2s0f0
UUID=71b3c94b-e746-44f8-a2e8-21b452746dba
DEVICE=enp2s0f0
ONBOOT=yes
IPADDR=10.174.28.173
PREFIX=22
GATEWAY=10.174.28.1
DNS1=10.129.2.34
IPV6_PRIVACY=no
```

说明

- BOOTPROTO表示设备的IP类型，如果是静态IP可设置成static或none；如果是动态IP，请配置成dhcp，自动获取IP。
- 更改ONBOOT参数为yes，设置自动启动网络连接。
- PREFIX表示网络位，值为22对应的子网掩码为255.255.252.0。

步骤4 重启网络服务。

```
service network restart
```

步骤5 查看IP地址是否配置成功。

```
ifconfig
```

步骤6 执行以下命令查看服务状态。

```
systemctl status NetworkManager
```

若显示为disabled，则执行以下命令将NetworkManager服务设置为自启动。

```
systemctl enable NetworkManager
```

再次查看服务状态，若为enabled则配置成功。

----结束

Ubuntu 操作系统配置网卡 IP 地址

步骤1 以root用户进入OS界面。

步骤2 进入网卡的配置文件，此处以网卡名为enp125s0f0举例，具体网卡请以实际环境为准。

```
vi /etc/netplan/01-netcfg.yaml
```

步骤3 配置业务IP地址、子网掩码和网关，如[图A-2](#)所示。

图 A-2 配置 IP 地址

```
root@Taishan-2280-V2:/etc/netplan# cat 01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp125s0f0:
      dhcp4: no
      addresses: [10.174.216.162/23, ]
      gateway4: 10.174.216.1
root@Taishan-2280-V2:/etc/netplan#
```

步骤4 应用配置文件。

```
netplan apply
```

步骤5 查看IP地址是否配置成功。

```
ifconfig
```

----结束

A.6 设置用户有效期

为保证用户的安全性，应设置用户的有效期，使用系统命令chage来设置用户的有效期。

命令为：

```
chage [-m mindays] [-M maxdays] [-d lastday] [-I inactive] [-E expiredate] [-W warndays] user
```

相关参数请参见[表A-2](#)。

表 A-2 设置用户有效期

参数	参数说明
-d --lastday	上一次更改的日期。
-E --expiredate	用户到期的日期。超过该日期，此用户将不可用。
-h --help	显示命令帮助信息。
-i --iso8601	更改用户密码的过期日期并以YYYY-MM-DD格式显示。
-l --inactive	停滞时期。过期指定天数后，设定密码为失效状态。
-l --list	列出当前的设置。由非特权用户来确定口令或账户何时过期。
-m --mindays	口令可更改的最小天数。设置为“0”表示任何时候都可以更改口令。
-M --maxdays	口令保持有效的最大天数。设置为“-1”表示可删除这项口令的检测。设置为“99999”，表示无限期。
-R --root	将命令执行的根目录设置为指定目录。
-W --warndays	用户口令到期前，提前收到警告信息的天数。

 说明

- 日期格式为YYYY-MM-DD，如chage -E 2017-12-01 test表示用户test的口令在2017年12月1日过期。
- User必须填写，填写时请替换为具体用户，默认为root用户。

举例说明：修改用户test的有效期为90天。

```
chage -M 90 test
```

A.7 配置 pip 源

配置pip源，配置方法如下：

步骤1 使用软件包的安装用户，执行如下命令：

```
cd ~/.pip
```

如果提示目录不存在，则执行如下命令创建：

```
mkdir ~/.pip  
cd ~/.pip
```

步骤2 编辑pip.conf文件。

使用**vi pip.conf**命令打开pip.conf文件，写入如下内容：

```
[global]  
#以华为源为例，请根据实际情况进行替换。  
index-url = https://mirrors.huaweicloud.com/repository/pypi/simple  
trusted-host = mirrors.huaweicloud.com  
timeout = 120
```

步骤3 执行**wq!**命令保存文件。

----结束

A.8 查询软件包版本信息

以下操作适用于查询CANN软件包（如toolkit等）的版本信息。

步骤1 以软件包的安装用户登录软件包的安装环境。

步骤2 进入软件包安装信息文件目录。（以下以Ascend-cann-toolkit软件包为例，其他软件包以实际目录为准）

```
cd /usr/local/Ascend/ascend-toolkit/latest/{arch}-linux
```

其中/usr/local/Ascend为root用户默认安装路径，请用户根据实际安装路径替换。
{arch}表示CPU架构（aarch64或x86_64）。

步骤3 执行以下命令获取版本信息。

```
cat ascend_toolkit_install.info
```

----结束

B FAQ

B.1 安装驱动时提示“The user and group are not same with last installation”

问题描述

安装驱动时提示：

```
The user and group are not same with last installation,do not support overwriting installation
```

可能原因

当前指定的驱动运行用户与“/etc/ascend_install.info”文件中记录的运行用户不一致，导致校验失败，安装退出。原因是之前卸载驱动和CANN软件时，aicpu没有被卸载，导致“/etc/ascend_install.info”文件未清空。

解决措施

先卸载aicpu，再安装驱动。卸载aicpu操作如下：

步骤1 以root用户登录安装环境。

步骤2 进入卸载脚本所在目录。

```
cd /usr/local/Ascend/opp/aicpu/script
```

步骤3 执行./uninstall.sh命令运行脚本，完成卸载。

----结束

B.2 openssl-devel 安装时报错提示 so 文件冲突

问题描述

运行yum install -y openssl-devel命令进行安装时，出现so文件冲突错误，如下图所示：

```
Downgrade 3 Packages
Total size: 5.1 M
Is this ok [y/N]: y
Downloading Packages:
(SKIPPED) krb5-libs-1.18-1.h2.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) libselinux-2.9-1.h4.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) python3-libselinux-2.9-1.h4.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) e2fsprogs-devel-1.45.6-0.h1.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) keyutils-libs-devel-1.6.1-1.h2.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) krb5-devel-1.18-1.h2.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) libselinux-devel-2.9-1.h4.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) libsepol-devel-2.9-1.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) libverto-devel-0.3.1-2.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) openssl-SMx-devel-1.1.1f-7.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) openssl-SMx-libs-1.1.1f-7.eulerosv2r9.aarch64.rpm: Already downloaded
(SKIPPED) pcre2-devel-10.33-2.h1.eulerosv2r9.aarch64.rpm: Already downloaded
Running transaction check
Transaction check succeeded.
The downloaded packages were saved in cache until the next successful transaction.
You can remove cached packages by executing 'yum clean packages'.
Error: Transaction test error:
  file /usr/lib64/libcrypto.so.1.1.1f.hmac from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/libssl.so.1.1.1f.hmac from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/engines-1.1/afalg.so from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/engines-1.1/capi.so from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/engines-1.1/padlock.so from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/libcrypto.so.1.1.1f from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
  file /usr/lib64/libssl.so.1.1.1f from install of openssl-SMx-libs-1:1.1.1f-7.eulerosv2r9.aarch64 conflicts with file from package openssl-libs-1:1.1.1f-7.h1.eulerosv2r9.aarch64
root@localhost lib64#
```

可能原因

原生包openssl-libs-1.1.1f-7.h1.eulerosv2r9 和即将安装的openssl-SMx-libs-1.1.1f-7.eulerosv2r9 包冲突。

注意：不要去强制卸载openssl-libs,因为ssl认证的服务依赖这个包，比如scp或者ssh等服务依赖于这个包的libcrypto.so.1.1库。另外这个包丢失，会导致rpm包失效。

解决措施

步骤1 新创建一个文件夹并进入。

```
mkdir openDown
cd openDown
```

步骤2 下载openssl的rpm包，通过以下命令下载到当前文件夹。

```
yum install --downloadonly --downloadaddr=. openssl-devel
```

步骤3 rpm命令强制安装。

```
rpm -Uvh *.rpm --force
```

----结束

B.3 pip3 安装软件包时，出现 read time out 报错

问题描述

使用pip3下载相关依赖报错，比如执行pip3 install scipy进行安装出现read time out 错误，如下图所示：

```
socket.timeout: The read operation timed out
During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/cli/base_command.py", line 188, in main
    status = self.run(options, args)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/commands/install.py", line 345, in run
    resolver.resolve(requirement_set)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/legacy_resolve.py", line 196, in resolve
    self._resolve_one(requirement_set, req)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/legacy_resolve.py", line 359, in _resolve_one
    abstract_dist = self._get_abstract_dist_for(req_to_install)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/legacy_resolve.py", line 307, in _get_abstract_dist_for
    self.require_hashes
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/operations/prepare.py", line 199, in prepare_linked_requirement
    progress_bar=self.progress_bar
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 1064, in unpack_url
    progress_bar=progress_bar
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 924, in unpack_http_url
    progress_bar)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 1152, in _download_http_url
    download_url(resp, link, content_file, hashes, progress_bar)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 861, in _download_url
    hashes.check_against_chunks(downloaded_chunks)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/utils/hashes.py", line 75, in check_against_chunks
    for chunk in chunks:
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 829, in written_chunks
    for chunk in chunks:
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/utils/ui.py", line 156, in iter
    for x in it:
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 818, in resp_read
    decode_content=False):
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/urllib3/response.py", line 531, in stream
    data = self.read(amt=amt, decode_content=decode_content)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/urllib3/response.py", line 496, in read
    raise IncompleteRead(self._fp_bytes_read, self.length_remaining)
  File "/usr/local/python3.7.5/lib/python3.7/contextlib.py", line 130, in __exit__
    self.gen.throw(type, value, traceback)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/urllib3/response.py", line 402, in _error_catcher
    raise ReadTimeoutError(self._pool, None, 'Read timed out.')
pip._vendor.urllib3.exceptions.ReadTimeoutError: HTTPConnectionPool(host='pypi.tuna.tsinghua.edu.cn', port=443): Read timed out.
```

可能原因

网络不稳定的情况下，下载时间长而且下载速度慢，尤其是下载 numpy和scipy这种较大的包的时候socket数据通信会经常出现这个问题。

解决措施

用 -i 参数来指定常用的其它源，如下命令所示：

```
pip3 install scipy -i xxxxxx
```

xxxxxx 为客户根据需求自行填写可用的源。

B.4 pip3 install scipy 报错

问题描述

安装scipy时，提示如下错误信息。

图 B-1 错误信息

```
root@localhost usr]# pip3.7 install scipy
Looking in indexes: http://mirrors.tools.huawei.com/pypi/simple
Collecting scipy
  Downloading http://mirrors.tools.huawei.com/pypi/packages/53/18/776750d57ade26522478a92a2e14035868624a6a62f4157b0cc5abdd4a980/scipy-1.5.2.tar.gz (25.4MB)
    [25.4MB 7.7MB/s]
Installing build dependencies ... done
Getting requirements to build wheel ... error
Preparing wheel metadata ... error
ERROR: Command errored out with exit status 1:
command: /usr/local/python3.7/bin/python3.7 /usr/local/python3.7/lib/python3.7/site-packages/pip/_vendor/pep517/_in_process.py prepare_metadata_for_build_wheel /tmp/tmp4vsvzlnw
cwd: /tmp/pip-install-qdp039wl/scipy
Complete output (253 lines):
lapack_opt_info:
lapack_mkl_info:
customize UnixCompiler
libraries mkl_rt not found in ['/usr/local/python3.7/lib', '/usr/local/lib64', '/usr/local/lib', '/usr/lib64', '/usr/lib', '/usr/lib/']
NOT AVAILABLE

openblas_lapack_info:
customize UnixCompiler
libraries openblas not found in ['/usr/local/python3.7/lib', '/usr/local/lib64', '/usr/local/lib', '/usr/lib64', '/usr/lib', '/usr/lib/']

Traceback (most recent call last):
  File "/usr/local/python3.7/lib/python3.7/site-packages/pip/_vendor/pep517/_in_process.py", line 207, in <module>
    main()
  File "/usr/local/python3.7/lib/python3.7/site-packages/pip/_vendor/pep517/_in_process.py", line 197, in main
    json_out['return_val'] = hook(**hook_input['kwargs'])
  File "/usr/local/python3.7/lib/python3.7/site-packages/pip/_vendor/pep517/_in_process.py", line 60, in prepare_metadata_for_build_wheel
    return hook(metadata_directory, config_settings)
  File "/tmp/pip-build-env-6g_9uv5/overlay/lib/python3.7/site-packages/setuptools/build_meta.py", line 157, in prepare_metadata_for_build_wheel
    self.run_setup()
  File "/tmp/pip-build-env-6g_9uv5/overlay/lib/python3.7/site-packages/setuptools/build_meta.py", line 249, in run_setup
    self.run_setup(setup_script=setup_script)
  File "/tmp/pip-build-env-6g_9uv5/overlay/lib/python3.7/site-packages/setuptools/build_meta.py", line 142, in run_setup
    exec(compile(code, file, 'exec'), locals())
  File "setup.py", line 583, in <module>
    setup_package()
  File "setup.py", line 579, in setup_package
    setup(**metadata)
  File "/tmp/pip-build-env-6g_9uv5/overlay/lib/python3.7/site-packages/numpy/distutils/core.py", line 135, in setup
    config = configuration()
  File "setup.py", line 477, in configuration
    raise NotFoundError(msg)
numpy.distutils.system_info.NotFoundError: No lapack/blas resources found.
```

可能原因

报错信息提示在/usr/lib64目录下找不到lapack和blas的库，但是实际在该目录下能找到，应该是识别不到so.3的库。

```
root@localhost usr]# find -name liblapack*
/lib64/liblapacke.so.3.8
/lib64/liblapack.so.3.8
/lib64/liblapack.so.3.8.0
/lib64/liblapacke.so.3
/lib64/liblapack.so.3
/lib64/liblapacke.so.3.8.0
root@localhost usr]# find -name libblas*
/lib64/libblas.so.3.8.0
/lib64/libblas.so.3.8
/lib64/libblas.so.3
```

解决措施

创建liblapack.so.3和libblas.so.3对应的so的软连接。

使用软件包的安装用户，执行如下命令：

```
cd /usr/lib64
ln -s libblas.so.3 libblas.so
ln -s liblapack.so.3 liblapack.so
```

B.5 pip3 install numpy 报错

问题描述

安装依赖时，使用**pip3 install numpy**命令安装时报错“Could not build wheels for numpy which use PEP 517 and cannot be install directly”，提示信息如下：

```
numpy/core/src/umath/loops.c.src: In function 'LONG_right_shift':
numpy/core/src/umath/loops.c.src:694:47: internal compiler error: in expand_shift_1, at expmed.c:2245
    BINARY_LOOP_FAST(@type@, @type@, *out = npy_rshift@((in1, in2));
                                         ^
numpy/core/src/umath/fast_loop_macros.h:180:9: note: in definition of macro 'BASE_BINARY_LOOP_S_INP'
    op; \
    ^
numpy/core/src/umath/loops.c.src:694:5: note: in expansion of macro 'BINARY_LOOP_FAST'
    BINARY_LOOP_FAST(@type@, @type@, *out = npy_rshift@((in1, in2));
    ^
Please submit a full bug report,
with preprocessed source if appropriate.
See <http://bugzilla.redhat.com/bugzilla> for instructions.
Preprocessed source stored into /tmp/ckpTHdy.out file, please attach this to your bugreport.
Running from numpy source directory.
/usr/local/python3.7.5/lib/python3.7/distutils/dist.py:274: UserWarning: Unknown distribution option: 'define_macros'
    warnings.warn(msg)
error: Command "gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O3 -Wall -fPIC -DNPY_INTERNAL_BUILD=1 -DHAVE_CBLAS -D_LARGEFILE64_SOURCE=1 -DHAVE_CBLAS -Ibuild/src.linux-aarch64-3.7/numpy/core/src/umath -Ibuild/src.linux-aarch64-3.7/numpy/core/src/common -Inumpy/core/include -Ibuild/src.linux-aarch64-3.7/numpy/core/include/numpy -Inumpy/core/src/common -Inumpy/core/src/multiarray -Inumpy/core/src/umath -Inumpy/core/src/npysort -I/usr/local/python3.7.5/include/python3.7m -Ibuild/src.linux-aarch64-3.7/numpy/core/src/npymath -c build/src.linux-aarch64-3.7/numpy/core/src/umath/loops.c -o build/temp.linux-aarch64-3.7/build/src.linux-aarch64-3.7/build/src.linux-aarch64-3.7/numpy/core/src/umath/loops.o -std=c99" failed with exit status 1

-----
ERROR: Failed building wheel for numpy
Failed to build numpy
ERROR: Could not build wheels for numpy which use PEP 517 and cannot be installed directly
```

可能原因

centos等系统默认安装的gcc版本较低，导致numpy安装失败。

解决措施

执行如下命令安装：

```
export CFLAGS=-std=c99
pip3 install numpy==1.19.2
```

B.6 pip3 install 报错 “subprocess.CalledProcessError: Command '('lsb_release', '-a')' return non-zero exit status 1”

问题描述

安装依赖时，使用**pip3 install xxx**命令安装相关软件时报错

“subprocess.CalledProcessError: Command '('lsb_release', '-a')' return non-zero exit status 1”，提示信息如下：

```
root@debian:/home/work_env/Python-3.7.5# pip3.7.5 install attrs
ERROR: Exception:
Traceback (most recent call last):
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/cli/base_command.py", line 188, in main
    status = self.run(options, args)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/commands/install.py", line 286, in run
    with self._build_session(options) as session:
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/cli/base_command.py", line 108, in _build_session
    index_urls=self._get_index_urls(options),
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 559, in __init__
    self.headers["User-Agent"] = user_agent()
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_internal/download.py", line 144, in user_agent
    zip(["name", "version", "id"], distro.linux_distribution()),
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 122, in linux_distribution
    return _distro.linux_distribution(full_distribution_name)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 677, in linux_distribution
    self.version()
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 737, in version
    self.lsb_release_attr('release'),
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 899, in lsb_release_attr
    return self._lsb_release_info.get(attribute, '')
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 552, in __get__
    ret = obj._dict[self._fname] = self._f(obj)
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pip/_vendor/distro.py", line 1012, in _lsb_release_info
    stdout = subprocess.check_output(cmd, stderr=devnull)
  File "/usr/local/python3.7.5/lib/python3.7/subprocess.py", line 411, in check_output
    **kwargs).stdout
  File "/usr/local/python3.7.5/lib/python3.7/subprocess.py", line 512, in run
    output=stdout, stderr=stderr)
subprocess.CalledProcessError: Command '('lsb_release', '-a')' returned non-zero exit status 1.
```

可能原因

用户自行编译安装的python3.7.5在执行subprocess模块时，在执行lsb_release -a 时提示找不到lsb_release.py模块，用户自行编译安装的python3.7.5的lib路径是“/usr/local/python3.7.5/lib/python3.7/”，该路径下没有lsb_release.py模块，因此会报错。

解决措施

步骤1 查找缺失文件'lsb_release.py'，使用如下命令：

```
find / -name lsb_release
```

执行上述命令后，得到如下路径，以下仅为示例，用户实际可能有差别：

```
/usr/bin/lsb_release
```

步骤2 将**步骤1**找到的“/usr/bin/lsb_release”文件备份：

```
mv /usr/bin/lsb_release /usr/bin/lsb_release.bak
```

步骤3 然后执行**pip3 list**查看是否解决。

----结束

B.7 Python 版本不一致导致的问题

问题描述

普通用户下，如果用户的其他软件依赖的Python版本和我们要求的Python版本不一致，可能会出现問題。

解决措施

建议使用Python的venv。Python虚拟环境用于将软件包安装与系统隔离开来。

B.8 安装 run 包失败，日志中报错 “usergroup=root not right!”

问题描述

安装run包失败，日志中报错 “usergroup=root not right!”。

图 B-2 报错截图

```
Toolkit] [20230407-11:03:13] [INFO] start uninstall CANN-toolkit-1.81.22.7.220-linux.aarch64.run
Toolkit] [2023-04-07 11:03:24] [ERROR]: usergroup=root not right! Please check the relationship of paas and root
Toolkit] [2023-04-07 11:03:24] [INFO]: End Time: 2023-04-07 11:03:24
Toolkit] [20230407-11:03:25] [INFO] delete operation, the directory /usr/local/Ascend/ascend-toolkit is not empty.
Toolkit] [20230407-11:03:25] [ERROR] CANN-toolkit-1.81.22.7.220-linux.aarch64.run install failed
```

可能原因

使用非root用户登录环境，切换至root用户执行安装操作时，切换后系统环境变量 **USER** 值仍为登录环境的非root用户。

解决措施

执行如下命令，手动将 **USER** 环境变量设置为实际执行安装的用户后，再进行安装：

```
export USER=root
```

B.9 使用 glibc 2.28 版本运行业务调用 malloc 接口出现 Core Dump 问题

问题描述

在glibc 2.28版本的docker环境中运行视频解析、目标识别相关业务时出现Core Dump，通过排查堆栈信息，实际为调用glibc的malloc接口时出现异常，堆栈信息如下图所示。

图 B-3 堆栈信息

```
Missing separate debuginfo for /usr/local/iGET/SRE/software/VA2/bin/./lib/libcrypto_module.so
--Type <RET> for more, q to quit, c to continue without paging--
Core was generated by ./va_frame --gpu 0 --ods_addr 90.77.91.194 --ods_port 1158 --subcmu_dispatch_po'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  tcache_get (tc_idx=1) at malloc.c:2948
2948      tcache->entries[tc_idx] = e->next;
[Current thread is 1 (Thread 0xffffd897f43a0 (LWP 1182274))]
(gdb)
(gdb)
(gdb)
(gdb) bt
#0  tcache_get (tc_idx=1) at malloc.c:2948
#1  __GI___libc_malloc (bytes=40) at malloc.c:3057
#2  0x0000ffff830b03d4 in operator new(unsigned long) () from /usr/lib64/libstdc++.so.6
#3  0x0000ffff89ab4ca8 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#4  0x0000ffff89ab5704 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#5  0x0000ffff89ab5998 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#6  0x0000ffff89ab5c7c in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#7  0x0000ffff89ad3b44 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#8  0x0000ffff8999a600 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#9  0x0000ffff899c7380 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#10 0x0000ffff89a6ec14 in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#11 0x0000ffff89926cac in ?? () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#12 0x0000ffff89879728 in rtStreamCreateWithFlags () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#13 0x0000ffff89d75824 in aclrtCreateStream () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libascendcl.so
#14 0x0000ffff8eed8c58 in AclProxy::ProxyAclrtCreateStream(void*, unsigned long) () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#15 0x0000ffff8eedd0f4 in AscendDecoder::CreatePicDecoder() () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#16 0x0000ffff8eede1f0 in VT_PicDecOpen () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#17 0x0000ffff90028d80 in CVideoInputImage::image_decode_thread(void*) () from /usr/local/Ascend/nnrt/latest/aarch64-linux/lib64/libruntime.so
#18 0x0000ffff9118d7b0 in start_thread (arg=0xffffe49fe52a0) at pthread_create.c:486
#19 0x0000ffff82e7c22c in thread_start () at ../sysdeps/unix/sysv/linux/aarch64/clone.S:80
(gdb) info line
line 2948 of "malloc.c" starts at address 0xffff82e210ac <_int_malloc+2444> and ends at 0xffff82e210b4 <_int_malloc+2452>.
(gdb)
```

解决措施

因glibc 2.28版本存在bug引发该问题，glibc 2.29版本现已修复，用户可以在执行业务之前，通过设置环境变量的方式修改malloc接口，配置命令如下：

```
export GLIBC_TUNABLES=glibc.malloc.tcache_count=0
```

B.10 openEuler 22.03 上运行业务时，出现 firewalld 相关 soft lockup 或 ksoftirqd 占用 CPU 过高

问题描述

在openEuler 22.03上同时使用默认的18.09版本docker与1.0.2版本firewalld时，运行业务出现firewalld相关soft lockup或ksoftirqd占用CPU过高的情况。

查询结果如下所示：

图 B-4 查询结果 1

```
top - 10:04:39 up 16:40, 5 users, load average: 3.58, 3.35, 3.11
Tasks: 2020 total, 2 running, 2018 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.4 id, 0.0 wa, 0.0 hi, 0.5 si, 0.0 st
MiB Mem : 770865.1 total, 691639.8 free, 8959.9 used, 70265.4 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used, 756585.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
588	root	20	0	0	0	0	R	99.3	0.0	363:18.21	ksoftirqd/114

图 B-5 查询结果 2

```
Jun 15 03:15:53 localhost kernel: [35477.382201] watchdog: BUG: soft lockup - CPU#166 stuck for 67s! [firewalld:1376895]
```

可能原因

18.09版本docker基于iptables写入规则，1.0.2版本firewalld基于nftables写入规则，两者无法匹配。

解决措施

编辑“/etc/firewalld/firewalld.conf”文件，将FirewallBackend字段修改为iptables。

B.11 安装原生 PyTorch 框架时使用华为源下载 typing 依赖导致 Python 环境错误

问题描述

在pip设置为华为源时，安装requirements.txt中的typing依赖后，会导致Python环境错误。

解决措施

在pip设置为华为源时，需打开requirements.txt文件，删除typing依赖，再执行命令。

```
vi requirements.txt #进入requirements.txt删除typing依赖
pip3 install -r requirements.txt #重新安装依赖列表
```

B.12 PyTorch 框架在 ARM CPU 上算子计算结果异常

问题描述

当前使用的PyTorch官方原生框架在ARM CPU上运行时，算子计算结果会出现异常，此问题为原生框架社区的已知问题，详细内容可参考PyTorch官方社区[ISSUE](#)。

解决措施

可通过以下方式解决：

- 修改算子输入数据类型，使用float64数据类型进行运算。
- 升级编译ARM版本PyTorch使用的gcc编译器至9.4版本及以上，并使用相同编译器重新编译torch_npu、APEX、MMCV等其他配套软件（避免因编译器版本不匹配导致兼容性问题）。
- 在Docker环境编译torch_npu。

- a. 拉取容器镜像。

```
docker pull quay.io/pypa/manylinux2014_aarch64:latest
```

- b. 执行命令查看镜像信息。

```
docker images
```

回显如下：

REPOSITORY	IMAGE ID	CREATED	SIZE
quay.io/pypa/manylinux2014_aarch64			
latest	fe2afc7b4b0d	9 days ago	2.09GB

- c. 启动镜像并在后台运行。

```
docker run -dit [IMAGE ID] bash
```

[IMAGE ID]为**b**查询出的镜像ID。

- d. 执行命令查看运行的容器。

```
docker ps
```

回显如下：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
6c2ead81ede5	fe2afc7b4b0d	"manylinux-entrypoin..."	6 seconds ago	Up 4	
	seconds	dreamy_einstein			

- e. 执行命令进入容器。

```
docker exec -it [CONTAINER ID] bash
```

[CONTAINER ID]为**d**查询到的容器ID。

- f. 编译torch_npu。

- i. 创建软链接。

```
cd /usr/local/bin/
ln -s /opt/_internal/cpython-3.7.17/bin/pip3.7 pip3.7
ln -s /opt/_internal/cpython-3.8.17/bin/pip3.8 pip3.8
ln -s /opt/_internal/cpython-3.9.17/bin/pip3.9 pip3.9
ln -s python3.7 python3
```

- ii. 安装PyTorch 1.11.0以及依赖。

```
pip3.7 install torch==1.11.0
pip3.7 install pyyaml
```

- iii. 进入任意目录，下载torch_npu源码，此处以“/home/”为例。

```
cd /home/  
git clone https://gitee.com/ascend/pytorch.git -b v1.11.0 --depth=1 v1.11.0  
cd v1.11.0/
```
- iv. 编译torch_npu。

```
bash ci/build.sh --python=3.7
```
- g. 退出容器，在服务器取回编译完成的包并安装。

```
exit  
docker cp [CONTAINER ID]:/home/v1.11.0/dist/torch_npu-1.11.0.post1-cp37-cp37m-  
linux_aarch64.whl ./  
pip3 install torch_npu-1.11.0.post1-cp37-cp37m-linux_aarch64.whl
```

B.13 原生 PyTorch 编译过程报错 no module named yaml/typing_extensions.

问题描述

PyTorch编译依赖yaml库和typing_extensions库，需要手动安装。

解决措施

安装依赖成功后，需要先执行**make clean**再执行**bash build.sh**进行编译，否则可能因缓存出现未知编译错误。

```
pip3 install pyyaml  
pip3 install typing_extensions
```

B.14 PyTorch 运行遇到找不到 te 问题

问题描述

PyTorch运行过程中，提示缺少te依赖。

解决措施

在开发环境下：

```
cd /usr/local/Ascend/ascend-toolkit/latest/{arch}-linux/lib64 # {arch}为架构名称。  
pip3 install --upgrade topi-0.4.0-py3-none-any.whl  
pip3 install --upgrade te-0.4.0-py3-none-any.whl
```

在运行环境下：

```
cd /usr/local/Ascend/nnae/latest/{arch}-linux/lib64 # {arch}为架构名称。  
pip3 install --upgrade topi-0.4.0-py3-none-any.whl  
pip3 install --upgrade te-0.4.0-py3-none-any.whl
```

B.15 PyTorch 编译过程提示 CMAKE 相关错误

问题描述

PyTorch编译时cmake依赖时提示找不到包、编译cmake报错版本过低，可使用安装脚本或源码编译安装。

解决措施

方法一：下载安装脚本安装cmake。（参考[cmake官网](#)）

X86_64环境脚本安装：cmake-3.12.0-Linux-x86_64.sh

aarch64环境脚本安装：cmake-3.12.0-Linux-aarch64.sh

步骤1 执行命令。

```
./cmake-3.12.0-Linux-{arch}.sh #{arch}为架构名称
```

步骤2 设置软连接。

```
ln -s /usr/local/cmake/bin/cmake /usr/bin/cmake
```

步骤3 执行如下命令验证是否安装成功。

```
cmake --version
```

如显示“cmake version 3.12.0”则表示安装成功。

----结束

方法二：使用源码编译安装。

步骤1 获取cmake软件包。

```
wget https://cmake.org/files/v3.12/cmake-3.12.0.tar.gz --no-check-certificate
```

步骤2 解压并进入软件包目录。

```
tar -xf cmake-3.12.0.tar.gz
cd cmake-3.12.0/
```

步骤3 执行配置、编译和安装命令。

```
./configure --prefix=/usr/local/cmake
make && make install
```

步骤4 设置软连接。

```
ln -s /usr/local/cmake/bin/cmake /usr/bin/cmake
```

步骤5 执行如下命令验证是否安装成功。

```
cmake --version
```

如显示“cmake version 3.12.0”则表示安装成功。

----结束

B.16 PyTorch 运行提示找不到 libblas.so

问题描述

环境缺少openblas库，需要安装openblas库。

解决措施

在CentOS、EulerOS环境下：

```
yum -y install openblas
```

在Ubuntu环境下：

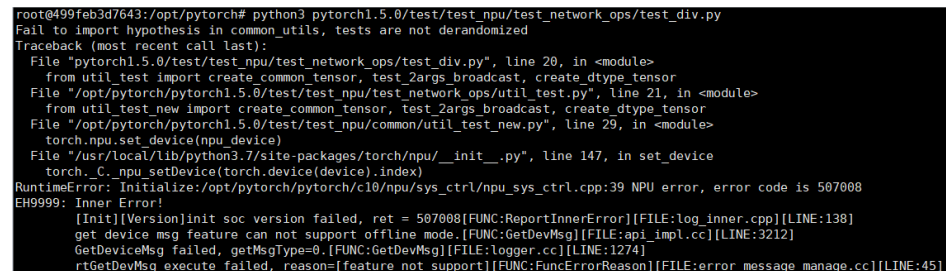
```
apt install libopenblas-dev
```

B.17 PyTorch 在容器中运行未挂载 device 问题

问题描述

在容器中运行脚本出现NPU相关ERROR。由于启动容器实例时，未挂载device参数，导致无法正常启动实例。

图 B-6 错误提示



```
root@499feb3d7643:/opt/pytorch# python3 pytorch1.5.0/test/test_npu/test_network_ops/test_div.py
Fail to import hypothesis in common utils, tests are not derandomized
Traceback (most recent call last):
  File "/opt/pytorch/pytorch1.5.0/test/test_npu/test_network_ops/test_div.py", line 20, in <module>
    from util_test import create_common_tensor, test_2args_broadcast, create_dtype_tensor
  File "/opt/pytorch/pytorch1.5.0/test/test_npu/test_network_ops/util_test.py", line 21, in <module>
    from util_test_new import create_common_tensor, test_2args_broadcast, create_dtype_tensor
  File "/opt/pytorch/pytorch1.5.0/test/test_npu/common/util_test_new.py", line 29, in <module>
    torch.npu.set_device(npu_device)
  File "/usr/local/lib/python3.7/site-packages/torch/npu/__init__.py", line 147, in set_device
    torch._C._npu_setDevice(torch.device(device).index)
RuntimeError: Initialize:/opt/pytorch/pytorch/c10/npu/sys_ctrl/npu_sys_ctrl.cpp:39 NPU error, error code is 507008
EH9999: Inner Error!
[Init][Version]init soc version failed, ret = 507008[FUNC:ReportInnerError][FILE:log_inner.cpp][LINE:138]
get device msg feature can not support offline mode.[FUNC:GetDevMsg][FILE:api_impl.cc][LINE:3212]
GetDeviceMsg failed, getMsgType=0.[FUNC:GetDevMsg][FILE:logger.cc][LINE:1274]
rtGetDevMsg execute failed, reason=[feature not support][FUNC:FuncErrorReason][FILE:error_message_manage.cc][LINE:45]
```

解决措施

请用户参考以下命令，重启容器。

```
docker run -it --ipc=host \
--device=/dev/davinciX \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /usr/local/Ascend/driver \
-v /usr/local/dcmi \
-v /usr/local/bin/npu-smi \
${镜像名称}:${tag} \
/bin/bash
```

参数说明：

- /dev/davinciX: NPU设备，X是芯片物理ID号，例如davinci0。
- /dev/davinci_manager: 管理设备。
- /dev/devmm_svm: 管理设备。
- /dev/hisi_hdc: 管理设备。
- /usr/local/Ascend/driver: 驱动目录。
- /usr/local/dcmi: DCMi目录。
- /usr/local/bin/npu-smi: npu-smi工具。
- \${镜像名称}:\${tag}: 镜像名称与版本号。

B.18 PyTorch 运行 import torch_npu 显示 _has_compatible_shallow_copy_type 重复注册 warning 问题

问题描述

warning如下图所示，由Tensor.set_data浅拷贝操作触发。主要原因是PyTorch插件化并解耦后，“_has_compatible_shallow_copy_type”缺乏对NPU Tensor的浅拷贝判断支持，因此需要重新注册“_has_compatible_shallow_copy_type”。

图 B-7 错误提示

```
>>> import torch_npu
[W OperatorEntry.cpp:121] Warning: Overriding a previously registered kernel for the same operator and the same dispatch key
operator: aten::_has_compatible_shallow_copy_type(Tensor self, Tensor from) -> (bool)
  registered at /home/programs/device/ /build/aten/src/ATen/RegisterSchema.cpp:20
dispatch key: Math
previous kernel: registered at /home/programs/device/ /build/aten/src/ATen/RegisterMath.cpp:5686
new kernel: registered at /home/programs/device/ /torch_npu/csrc/aten/ops/HasCompatibleShallowCopyType.cpp:37 (function registerKernel)
```

解决措施

该warning不影响模型的精度和性能，可以忽略。待NPU设备号合入社区或者后续PyTorch版本“_has_compatible_shallow_copy_type”注册方式发生变动，该warning会被解决。

B.19 PyTorch 运行 import torch_npu 显示 No module named 'torch_npu._C'

问题描述

在编译torch_npu的目录进入Python引用torch_npu报错问题。

图 B-8 报错提示

```
root@ubuntu:/home/test_build_pt1.8/pytorch# python3
Python 3.7.5 (default, May 5 2022, 18:11:22)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.8.1+cpu'
>>> import torch_npu
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import torch_npu._C
  File "/home/test_build_pt1.8/pytorch/torch_npu/_init_.py", line 26, in <module>
    import torch_npu.npu
  File "/home/test_build_pt1.8/pytorch/torch_npu/npu/_init_.py", line 41, in <module>
    from .utils import (is_initialized, _lazy_call, _lazy_init, init, set_dump,
  File "/home/test_build_pt1.8/pytorch/torch_npu/npu/utils.py", line 27, in <module>
    import torch_npu._C
ModuleNotFoundError: No module named 'torch_npu._C'
>>>
```

解决措施

请切换至其他目录验证torch_npu的引入。在编译目录执行验证会在当前目录下查找torch_npu，故会报错。

B.20 PyTorch 编译时出现 Breakpad error: field 'regs' has incomplete type 'google_breakpad::user_regs_struct'报错

问题描述

PyTorch编译时出现Breakpad error: field 'regs' has incomplete type 'google_breakpad::user_regs_struct'报错。

解决措施

编译原生PyTorch时，未配置相关环境变量，导致编译不成功。

步骤1 执行命令配置环境变量。

```
export BUILD_BREAKPAD=0
```

步骤2 执行命令清除当前编译内容。

```
make clean
```

步骤3 重新编译。

----结束

C 附录

C.1 参数说明

软件包支持根据命令行完成一键式安装，各个命令之间可以配合使用，用户根据安装需要选择对应参数完成安装，所有参数都是可选参数。

安装命令格式： `./软件包名.run [options]`

详细参数请参见[表C-1](#)。

须知

如果通过 `./软件包名.run --help` 命令查询出的参数未解释在如下表格，则说明该参数预留或适用于其他芯片版本，用户无需关注。

表 C-1 安装包支持的参数说明

参数	说明
<code>--help -h</code>	查询帮助信息。
<code>--version</code>	查询版本信息。
<code>--info</code>	查询软件包构建信息。
<code>--list</code>	查询软件包文件列表。
<code>--check</code>	检查软件包的一致性和完整性。
<code>--quiet</code>	静默安装，跳过交互式信息。 使用此参数安装或升级时，将默认签署 华为企业业务最终用户许可协议（EULA） 。
<code>--nox11</code>	不使用x11模式运行。

参数	说明
--noexec	解压软件包到当前目录，但不执行安装脚本。配套--extract=<path>使用，格式为： --noexec --extract=<path>。
--extract=<path>	解压软件包中文件到指定目录。
--tar arg1 [arg2 ...]	对软件包执行tar命令，使用tar后面的参数作为命令的参数。例如执行--tar xvf命令，解压run安装包的内容到当前目录。
--install	安装软件包。后面可以指定安装路径--install-path=<path>，也可以不指定安装路径，直接安装到默认路径下。
--install-for-all	安装或升级时，允许其他用户具有安装群组的权限。当安装或者升级携带该参数时，软件包中创建的目录及文件，其他用户权限=安装群组权限。 该参数需要与--install、--devel、--upgrade等其中一个参数配合使用，例如./软件包名.run --install --install-for-all 说明 使用该参数将会存在安全风险：其他所有用户都有权限访问安装目录，请谨慎使用。
--install-path=<path>	指定安装路径，当环境上已存在全局配置文件“ascend_cann_install.info”时，该参数指定的路径必须与全局配置文件中保存的安装路径保持一致。如用户想更换安装路径，请删除全局配置文件“ascend_cann_install.info”或者修改全局配置文件内的安装路径为指定路径，同时建议卸载原路径下的CANN软件包。 可在如下目录查看是否存在该文件： • root用户：/etc/Ascend • 非root用户：\${HOME}/Ascend 若不指定，将安装到默认路径下： • 若使用root用户安装，默认安装路径为：/usr/local/Ascend。 • 若使用非root用户安装，则默认安装路径为：\${HOME}/Ascend。 若通过该参数指定了安装目录，root用户需要对指定的安装路径有755权限，非root用户需要750权限。
--force	强制安装、升级命令。当安装、升级检测不通过导致安装或升级失败时，则会提示用户使用该参数跳过安装检测，使用后方可安装或升级成功。 该参数需要配合“--install”或“--upgrade”一起使用。

参数	说明
--full	仅软件包toolkit支持使用该参数。 支持在无驱动场景下实现全量安装。
--uninstall	卸载已安装的软件。
--upgrade	升级已安装的软件，支持在低版本升级至高版本场景下使用。 如果需从高版本回退至低版本，需卸载高版本后重新安装所需版本。
--feature-list=<feature>	仅软件包toolkit、nnae支持使用该参数。 指定升级特性。仅支持升级使用，参数仅支持输入“Acclibs”（独立升级算子包），如“--feature-list=Acclibs”。 该参数需要配合“--upgrade”一起使用。 说明 推荐用户全量升级。如果版本配套表中有算子包独立升级的兼容性列表，则参考该兼容性列表，否则推荐用户全量升级。
--devel	按照开发模式安装软件包，即只安装开发环境需安装的文件。
--chip=<chip_type>	指定处理器类型，以便在安装过程中选择匹配的软件（如AICPU算子包）。 须知 该参数仅对软件包toolkit、nnrt、nnae有效。未指定此参数时，默认安装软件包内的全部AICPU算子包。
--alternative	仅软件包toolkit支持使用该参数。 查询可选安装特性列表。用于可选安装场景，在安装前查询软件包支持的可选安装特性列表，以便拷贝到--whitelist输入参数中。 toolkit可选安装特性如下： • atc：用于支持模型编译、模型转换。 • devtools：主要为调测、仿真工具。选择安装该特性时请配套安装nnae。 • nnrt：用于支持离线推理场景。 • nnae：用于支持离线推理、在线推理、训练以及IR构图场景。
--whitelist=<feature_type>	仅软件包toolkit支持使用该参数。 可选安装白名单，用于在执行安装（--install）时指定部分可选安装特性。不支持在升级场景下使用。 该参数需要配合“--install”一起使用。

参数	说明
--type=<package_type>	仅软件包kernels支持使用该参数。 指定已安装的nnae或toolkit软件包类型，用于在执行安装（--install）时指定跟随安装的软件包（“nnae”、“toolkit”），以识别到具体的安装目录。该参数需要配合“--install”或“--upgrade”一起使用。

 说明

以下参数未展示在--help参数中，用户请勿直接使用。

- --xwin：使用xwin模式运行。
- --phase2：要求执行第二步动作。

C.2 相关信息记录路径

CANN软件包在安装过程中会生成相关配置、日志信息等，文件存放路径如表C-2所示。

其中 $\{arch\}$ 表示CPU架构， $\${HOME}$ 为当前用户目录。

表 C-2 信息记录路径

信息说明	路径
软件包安装详细日志路径	以软件包toolkit为例（以软件包默认安装路径进行说明，请根据实际替换）： • root用户：“/var/log/ascend_seclog/ascend_toolkit_install.log” • 非root用户：“$\\${HOME}$/var/log/ascend_seclog/ascend_toolkit_install.log”
安装后软件包版本、CPU架构和安装路径等信息的记录路径	以软件包toolkit（nnrt目录结构同toolkit）和nnae（tfplugin目录结构同nnae）为例（以软件包默认安装路径进行说明，请根据实际替换）： • root用户： toolkit：“/usr/local/Ascend/ascend-toolkit/latest/$\{arch\}$-linux/ascend_toolkit_install.info” nnae：“/usr/local/Ascend/nnae/latest/ascend_nnae_install.info” • 非root用户： toolkit：“$\\${HOME}$/Ascend/ascend-toolkit/latest/$\{arch\}$-linux/ascend_toolkit_install.info” nnae：“$\\${HOME}$/Ascend/nnae/latest/ascend_nnae_install.info”

信息说明	路径
软件包安装路径的记录路径	- root用户: “/etc/Ascend/ascend_cann_install.info” - 非root用户: “\${HOME}/Ascend/ascend_cann_install.info”
软件包安装时指定的安装参数（如--install-for-all、--whitelist等）的记录路径	以软件包toolkit（nnrt目录结构同toolkit）和nnae（tfplugin目录结构同nnae）为例（以软件包默认安装路径进行说明，请根据实际替换）： - root用户: toolkit: “/usr/local/Ascend/ascend-toolkit/latest/{arch}-linux/install.conf” nnae: “/usr/local/Ascend/nnae/latest/install.conf” - 非root用户: toolkit: “\${HOME}/Ascend/ascend-toolkit/latest/{arch}-linux/install.conf” nnae: “\${HOME}/Ascend/nnae/latest/install.conf”

C.3 AICPU Kernel 加载说明

表 C-3 AICPU Kernel 加载说明

场景	说明	约束
物理机	AICPU Kernel安装后跟随业务进程初始化，自动加载到Device。	- 不支持一个昇腾AI处理器同时被多个用户使用，各用户加载不同版本AICPU Kernel。 - 不支持在用户运行环境内，指定某个昇腾AI处理器加载特定版本的AICPU Kernel。

C.4 口令复杂度要求

口令至少满足如下要求：

- 口令长度至少8个字符。
- 口令必须包含如下至少两种字符的组合：
 - 一个小写字母
 - 一个大写字母
 - 一个数字

- 一个特殊字符：`~!@#\$%^&*()-_+=\|[]{};:'",<.>/?和空格
3. 口令不能和账号一样。

C.5 公网 URL

CANN源码涉及以下公网URL，主要用于下载华为相关软件包时的相关协议：

- <https://support.huawei.com/enterprise/zh/warranty-policy>
- <https://support.huawei.com/enterprise/en/warranty-policy>
- <https://www.huawei.com/cn/psirt/vul-response-process>
- <https://www.huawei.com/en/psirt/vul-response-process>