CANN 软件安装指南

文档版本	01
发布日期	2023-04-21

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

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

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

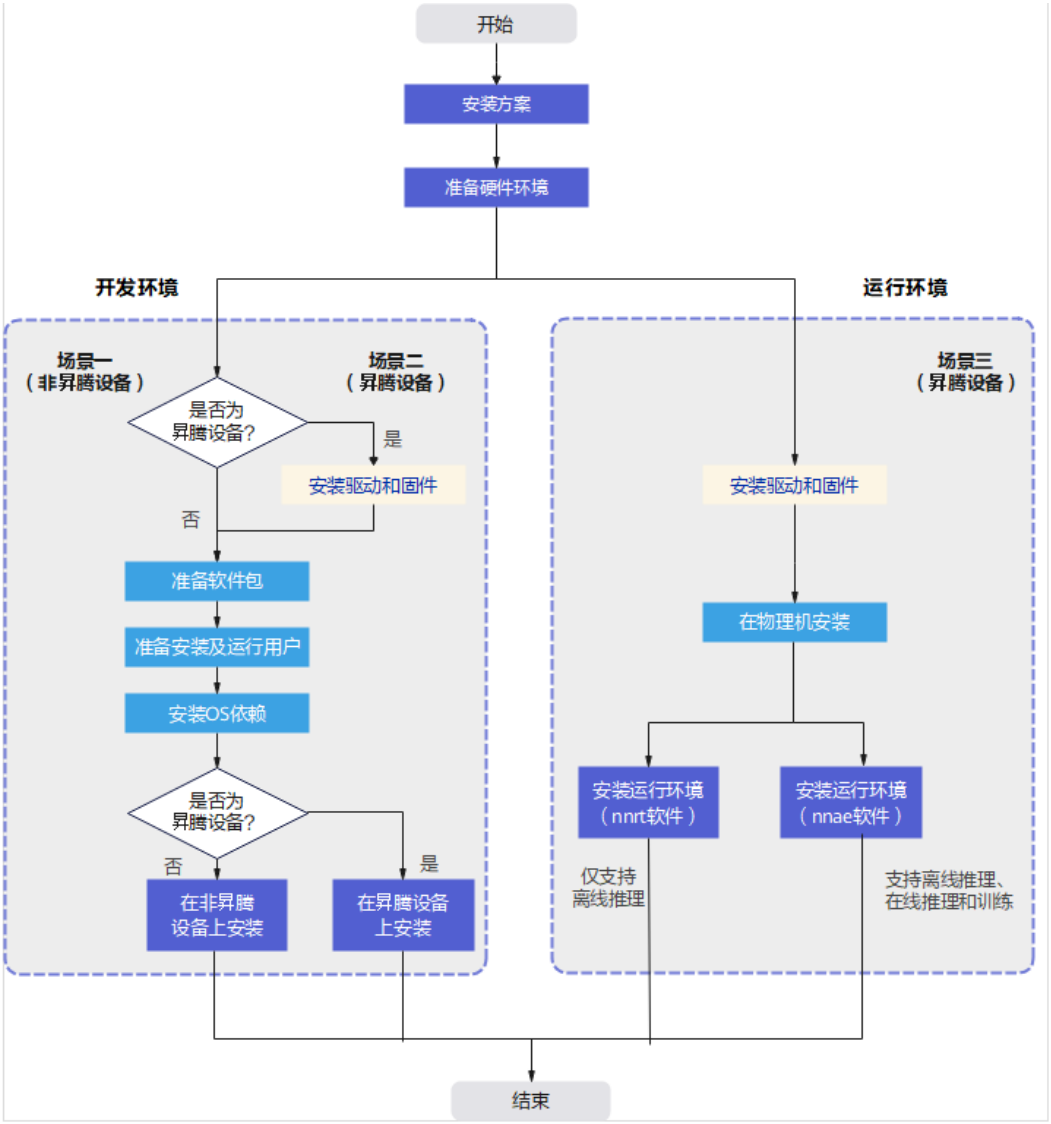
目 录

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。

- **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。**


开始
安装方案
准备硬件环境
开发环境
运行环境
场景一
（非昇腾设备）
场景二
（昇腾设备）
场景三
（昇腾设备）
是否为
昇腾设备?
是
安装驱动和固件
否
准备软件包
准备安装及运行用户
安装OS依赖
是否为
昇腾设备?
否
是
在非昇腾
设备上安装
在昇腾设备
上安装
安装驱动和固件
在物理机安装
安装运行环境
（nnrt软件）
安装运行环境
（nnae软件）
仅支持
离线推理
支持离线推理、
在线推理和训练
结束

2 安装方案

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。

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工具等。

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，开发环境上需同时部署x86_64和aarch64架构的开发套件，后续编译应用时需要调用aarch64架构开发套件所需的库文件。

2.2 Atlas 200（EP 场景）

Atlas 200（Atlas 200 AI加速模块）以昇腾310 AI 处理器的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）场景；如果PCIe工作在主模式，可以扩展外设，则称为RC（Root Complex）场景。具体可参考**C.3 昇腾产品形态说明**。

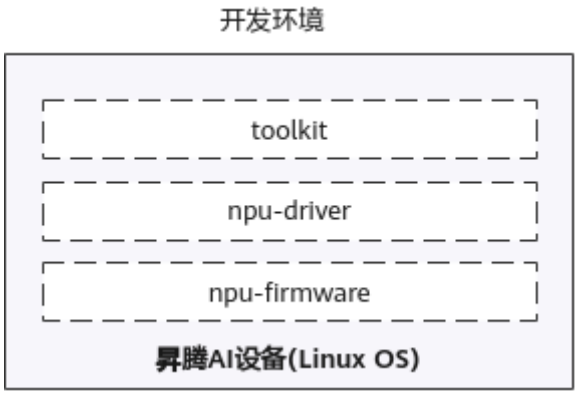
- 开发环境

  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

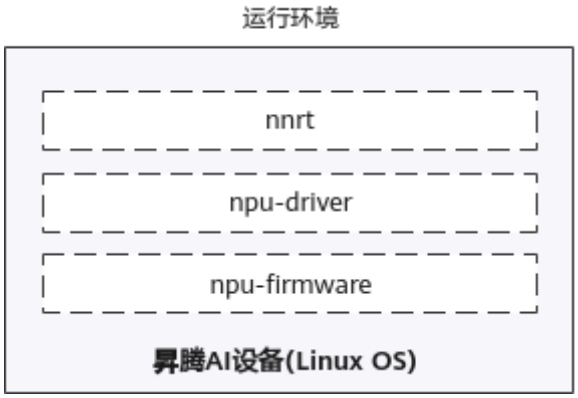
- 运行环境

  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。
- 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。

2.3 Atlas 200（RC 场景）

Atlas 200（Atlas 200 AI加速模块）以昇腾310 AI 处理器的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）场景；如果PCIe工作在从模式，则称为EP（Endpoint）场景。具体可参考**C.3 昇腾产品形态说明**。

- 开发环境

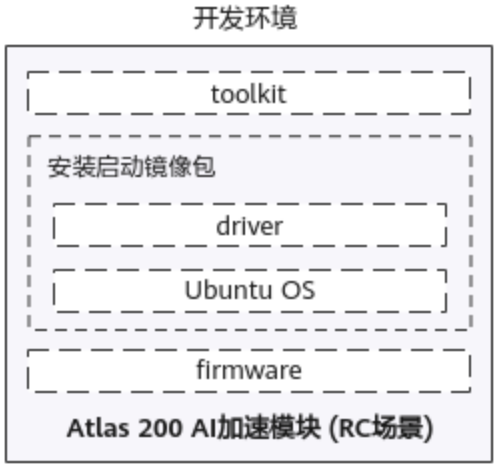
  安装示意图如**图2-4**所示。

  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库文件和开发工具等。

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加速模块运行环境安装具体请参考《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

2.4 Atlas 200I SoC A1

Atlas 200I SoC A1（Atlas 200I SoC A1核心板）开发或运行环境安装参考如下所示。

- 开发环境

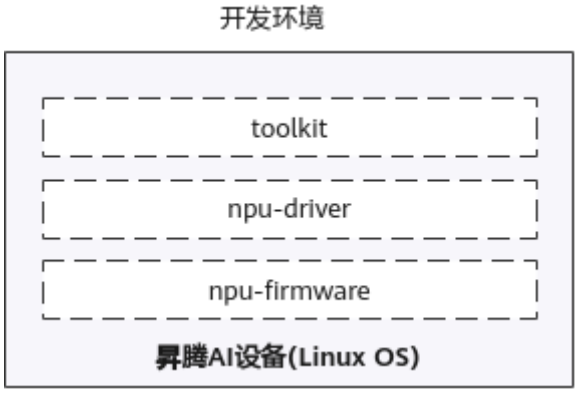
  开发环境安装软件如图2-6所示。

  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。

图 2-6 开发环境



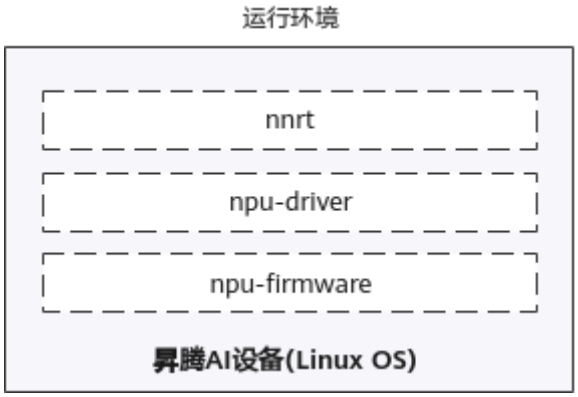
- 运行环境

  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。
- 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。

2.5 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：

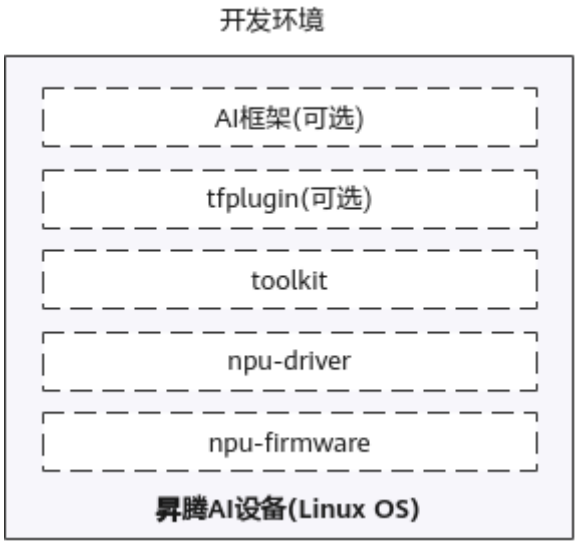
- 开发环境

  开发环境安装软件如图2-8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。
  - 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：如需进行在线推理，则需安装AI框架。

  图 2-8 开发环境

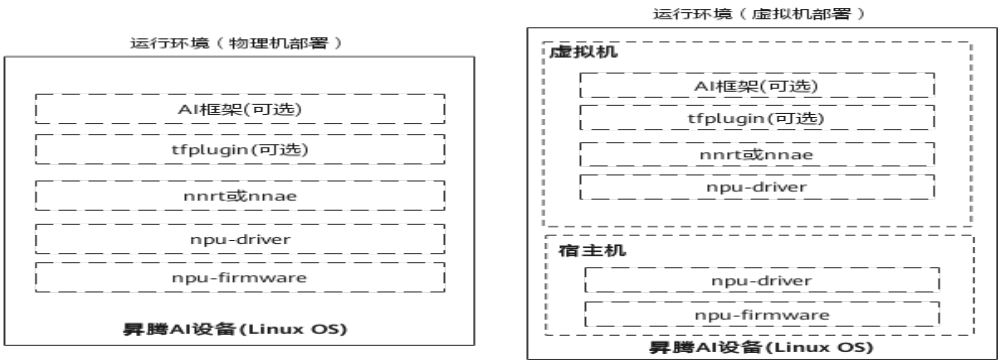
- 运行环境

  运行环境安装软件如图2-9所示。

  软件包说明：

  - npu-firmware：固件安装包。
  - npu-driver：驱动安装包。
  - 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，智能边缘（Atlas Intelligent Edge Solution）场景安装该软件包。
  - nnae：深度学习引擎包。支持离线推理、在线推理，数据中心（Ascend Data Center Solution）场景安装该软件包。
  - 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：如需进行在线推理，则需安装AI框架。

图 2-9 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。

2.6 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：

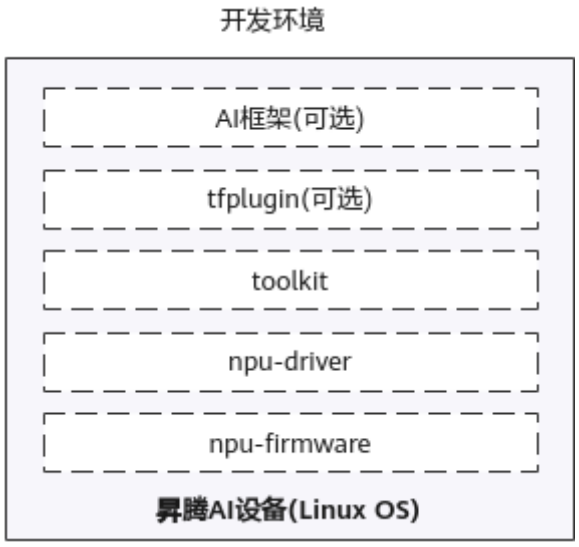
- 开发环境

  开发环境安装软件如图2-10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。
  - 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：如需进行在线推理，则需安装AI框架。

图 2-10 开发环境



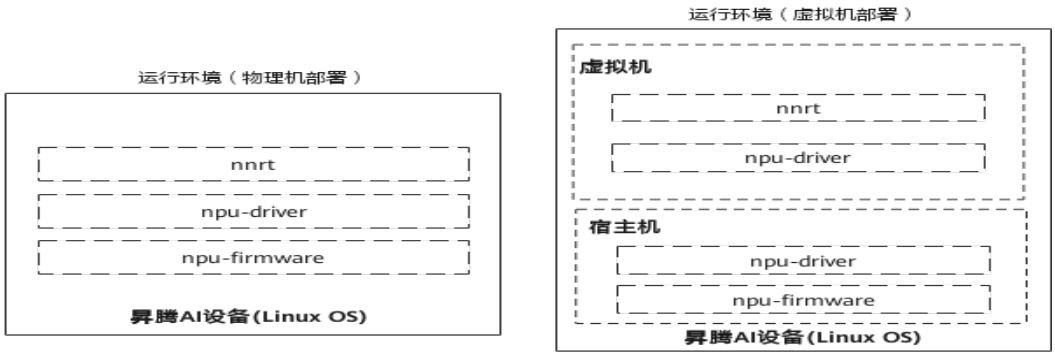
- 运行环境

  运行环境安装软件如图2-11所示。

  软件包说明：

  - npu-firmware：固件安装包。
  - npu-driver：驱动安装包。
  - nnae：深度学习引擎包。支持离线推理、在线推理。
  - 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：如需进行在线推理，则需安装AI框架。

图 2-11 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。

2.7 A500-3000

Atlas 500 智能小站（型号 3000）仅支持作为运行环境。

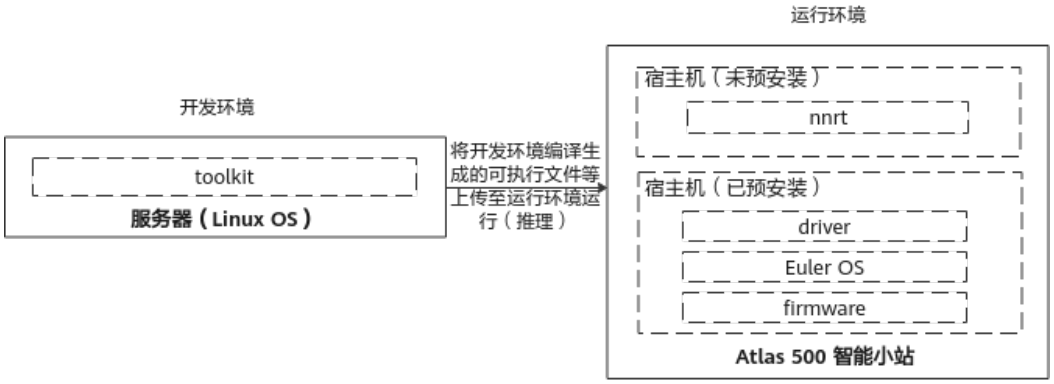
- 开发环境

  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

  Atlas 500 智能小站出厂预安装Euler OS、驱动&固件，Atlas 500 智能小站的初始配置请参考**《Atlas 500 智能小站 用户指南（型号 3000）》**。

安装方案如**图2-12**所示。

图 2-12 Atlas 500 AI edge station



2.8 A500 Pro-3000

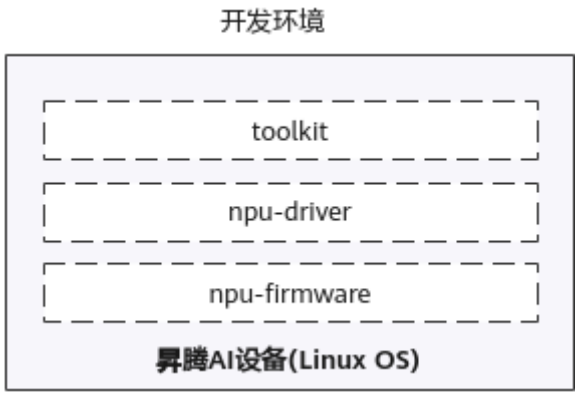
A500 Pro-3000（Atlas 500 Pro 智能边缘服务器 （型号 3000））开发或运行环境安装参考如下：

- 开发环境

  开发环境安装软件如**图2-13**所示。

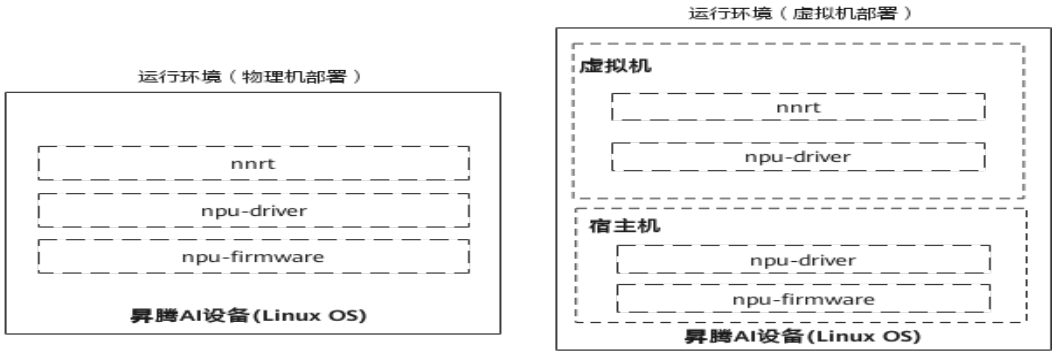
  软件包说明：
  - npu-firmware：固件安装包。
  - npu-driver：驱动安装包。
  - toolkit：开发套件包。主要用于用户开发应用、自定义算子和模型转换。开发套件包包含开发应用程序所需的AscendCL库文件、开发工具（如ATC模型转换工具）等。

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。
- 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。

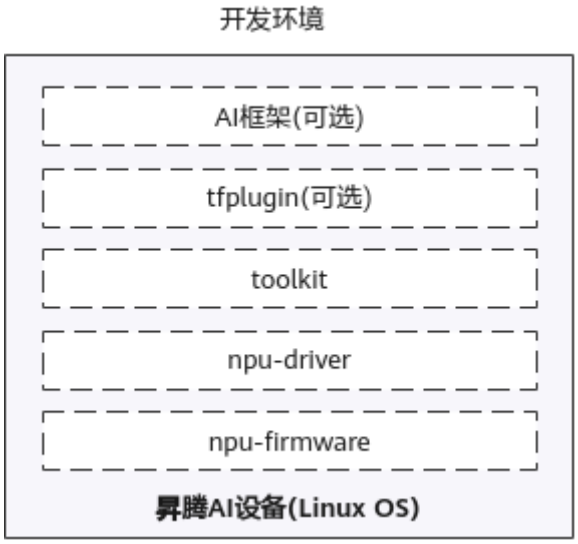
2.9 A800-3000、A800-3010

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2-15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。
- 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：如需进行在线推理，则需安装AI框架。

图 2-15 开发环境

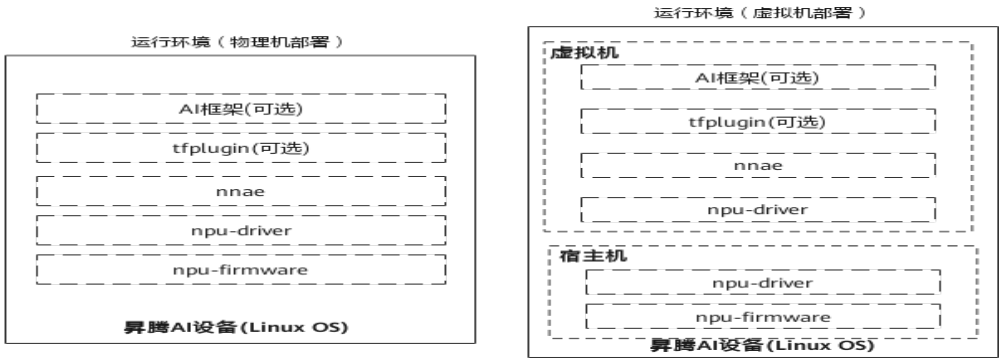


- 运行环境

  运行环境安装软件如图2-16所示，其中nnae软件包支持离线推理、在线推理。

  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、在线推理。
  - 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：如需进行在线推理，则需安装AI框架。

图 2-16 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。

2.10 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））以及A900-9000（Atlas 900 AI集群（型号 9000））开发或运行环境安装参考如下：

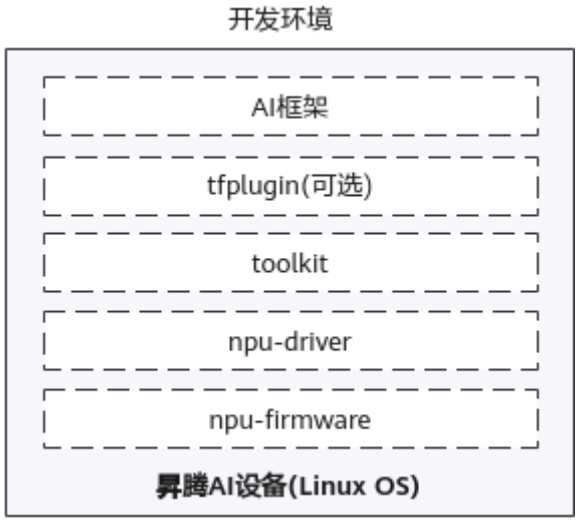
- 开发环境

  开发环境安装软件如图2-17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。
  - tfplugin：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 2-17 开发环境

- 运行环境

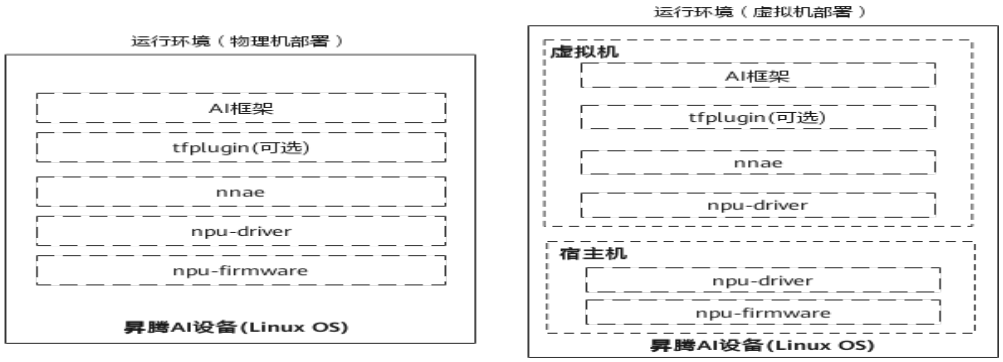
  运行环境安装软件如图2-18所示。

  软件包说明：

  - npu-firmware：固件安装包。

- npu-driver：驱动安装包。
- nnae：深度学习引擎包。支持离线推理、在线推理、训练。
- tfplugin：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 2-18 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

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 SoC A1	• Atlas 200I 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**》。

产品型号	参考手册
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**》。
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)**》。

产品型号	参考手册
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)**》。
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)**》。

说明

操作系统镜像请从官网获取，示例如下：

- Ubuntu 18.04.1：
  - aarch64

    从Ubuntu官网**http://old-releases.ubuntu.com/releases/18.04.1/**下载**ubuntu-18.04.1-server-arm64.iso**。
  - x86_64

    从Ubuntu官网**http://old-releases.ubuntu.com/releases/18.04.1/**下载**ubuntu-18.04.1-server-amd64.iso**。
- CentOS 7.6
  - aarch64

    从CentOS官网**https://archive.kernel.org/centos-vault/altarch/7.6.1810/isos/aarch64/**下载**CentOS-7-aarch64-Everything-1810.iso**。
  - x86_64

    从CentOS官网**http://vault.centos.org/7.6.1810/isos/x86_64/**下载**CentOS-7-x86_64-DVD-1810.iso**。

4 安装驱动和固件

请根据表4-1参考对应文档进行安装操作。

说明

- 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。
- 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 SoC A1	请参见《Atlas 200I SoC A1核心板 6.0.0 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《Ascend 310P 6.0.0 NPU驱动和固件安装指南（AI加速卡）》。
Atlas 300V Pro	请参见《Ascend 310P 6.0.0 NPU驱动和固件安装指南（AI加速卡）》。
Atlas 300V	请参见《Ascend 310P 6.0.0 NPU驱动和固件安装指南（AI加速卡）》。
Atlas 300I Duo	请参见《Ascend 310P 6.0.0 NPU驱动和固件安装指南（AI加速卡）》。
A300T-9000	请参见《Ascend 910 6.0.0 NPU驱动和固件安装指南（AI加速卡）》。

产品型号	参考文档
Atlas 300T Pro	请参见《**Ascend 910 6.0.0 NPU驱动和固件安装指南（AI加速卡）**》。
A500 Pro-3000	请参见《**Atlas 300I 推理卡 用户指南（型号 3000）**》的“安装与维护”章节。
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《**Ascend 910 6.0.0 NPU驱动和固件安装指南（AI服务器）**》。
A800-9010	请参见《**Ascend 910 6.0.0 NPU驱动和固件安装指南（AI服务器）**》。
A900-9000	请参见《**Ascend 910 6.0.0 NPU驱动和固件安装指南（AI服务器）**》。

5 CANN 软件包支持的 OS 清单

CANN软件包支持的OS清单如表5-1所示。

表 5-1 支持系统列表

产品型号	支持的操作系统
A800-9000	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.8、EulerOS 2.10 OpenEuler 20.03、OpenEuler 22.03 CentOS 7.6、CentOS 8.2 BCLinux 7.6、BCLinux 7.7 Kylin V10、Kylin V10 SP1 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
A800-3000+A300-3000	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.8、EulerOS 2.9、EulerOS 2.10 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

产品型号	支持的操作系统
A800-3000+A300-3010	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.8、EulerOS 2.9、EulerOS 2.10 OpenEuler 20.03、OpenEuler 22.03 CentOS 7.6、CentOS 8.2 Kylin V10 SP1 UOS20 1020e
A800-3000+A300T-9000 A800-3000+Atlas 300T Pro	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.8 OpenEuler 20.03、OpenEuler 22.03 CentOS 7.6、CentOS 8.2 Kylin V10 SP1 UOS20 1020e
A800-3000+Atlas 300I Pro A800-3000+Atlas 300V Pro	Ubuntu 18.04.5、Ubuntu 20.04 EulerOS 2.9、EulerOS 2.10 OpenEuler 20.03、OpenEuler 22.03 CentOS 7.6 Kylin V10 SP1
A800-3010+A300-3010	Ubuntu 18.04.1、Ubuntu 18.04.5、Ubuntu 16.04.5、Ubuntu 20.04 EulerOS 2.5、EulerOS 2.9、EulerOS 2.10 OpenEuler 20.03 CentOS 7.4、CentOS 7.6、CentOS 7.8、CentOS 8.2 SLES 12.4、SLES 12.5 BCLinux 7.6 Kylin V10 SP1
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
A800-3010+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 SLES 12.5 Kylin V10 SP1

产品型号	支持的操作系统
A800-3000 + Atlas 300V A800-3010 + Atlas 300V	Ubuntu 20.04 CentOS 7.8 OpenEuler22.03
A500 Pro-3000+A300 -3000	Ubuntu 16.04.5 EulerOS 2.8、EulerOS 2.9 OpenEuler 20.03 CentOS 7.6、CentOS 8.2 Linx 6.0 Kylin V10 SP1 UOS20 SP1
A500 Pro-3000+Atlas 300I Pro A500 Pro-3000+Atlas 300V Pro	Ubuntu 20.04 Linx 6.0.90、Linx 6.0.100 EulerOS 2.10 OpenEuler 20.03、OpenEuler 22.03 CentOS 7.6 Kylin V10 SP1
Atlas 200	RC：Ubuntu 18.04.4、Ubuntu 16.04.3 EP：Ubuntu 18.04.1
Atlas 200I Soc A1	OpenEuler 20.03
Atlas 300I Duo	Ubuntu 20.04 CentOS 7.8

6 安装开发环境

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。

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，需同时获取两种架构的开发套件包。	**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	（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。

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》，对从Support网站下载的软件包进行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.5 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，详情请参见**A.7 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 18.04、CentOS 7.6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、Linx系统可参考Ubuntu 18.04进行安装。
- EulerOS、OpenEuler、CentOS、BCLinux、Kylin、UOS20 1020e系统可参考CentOS 7.6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 18.04

表 6-2 依赖信息

类别	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）。
cmake	>=3.5.1
make	-
gcc	• 离线推理场景 要求4.8.5版本及以上gcc。 • 在线推理、训练、Ascend Graph开发场景 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

类别	版本要求
g++	装。若使用PyTorch 1.11.0，要求安装7.5.0及以上版本gcc。
zlib1g zlib1g-dev libsqlite3-dev openssl libssl-dev libffi-dev unzip pciutils net-tools libblas-dev gfortran libblas3 liblapack-dev liblapack3	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 absl-py	无版本要求，安装的版本以pip源为准。

CentOS 7.6

表 6-3 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）。

类别	版本限制
cmake	>=3.5.1
make	-
gcc gcc-c++	• 离线推理场景 要求4.8.5版本及以上gcc。 • 在线推理、训练、Ascend Graph开发场景 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。若使用PyTorch 1.11.0，要求安装7.5.0及以上版本gcc。
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools sqlite-devel lapack-devel gcc-gfortran	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 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）。
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 18.04）

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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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 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，请以实际情况为准）。

```
gcc (Ubuntu 7.3.0-3ubuntu1~18.04) 7.3.0
g++ (Ubuntu 7.3.0-3ubuntu1~18.04) 7.3.0
```



```
GNU Make 4.1
cmake version 3.10.2
zlib1g:arm64   1:1.2.11.dfsg-0ubuntu2 arm64       compression library - runtime
zlib1g-dev:arm64 1:1.2.11.dfsg-0ubuntu2 arm64       compression library - development
libsqlite3-dev:arm64 3.22.0-1ubuntu0.3 arm64       SQLite 3 development files
openssl  1.1.1-1ubuntu2.1~18.04.6 arm64   Secure Sockets Layer toolkit - cryptographic utility
libssl-dev:arm64 1.1.1-1ubuntu2.1~18.04.6 arm64     Secure Sockets Layer toolkit - development files
libffi-dev:arm64 3.2.1-8      arm64       Foreign Function Interface library (development files)
unzip          6.0-21ubuntu1 arm64       De-archiver for .zip files
pciutils       1:3.5.2-1ubuntu1 arm64       Linux PCI Utilities
net-tools      1.60+git20161116.90da8a0-1ubuntu1 arm64       NET-3 networking toolkit
libblas-dev:arm64 3.7.1-4ubuntu1 arm64       Basic Linear Algebra Subroutines 3, static library
gfortran       4:7.4.0-1ubuntu2.3 arm64       GNU Fortran 95 compiler
libblas3:arm64 3.7.1-4ubuntu1 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，则如下命令修改为只安装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 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**，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（3.7.0~ 3.7.11），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，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libsqlite3-dev依赖。

   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.8 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.7 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（CentOS 7.6）

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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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，修改线程最大数，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
```

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，请以实际情况为准）。


```
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39)
g++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39)
GNU Make 3.82
cmake version 2.8.12.2
unzip-6.0-21.el7.aarch64
zlib-devel-1.2.7-18.el7.aarch64
libffi-devel-3.0.13-18.el7.aarch64
openssl-devel-1.0.2k-19.el7.aarch64
pciutils-3.5.1-3.el7.aarch64
net-tools-2.0-0.25.20131004git.el7.aarch64
sqlite-devel-3.7.17-8.el7_7.1.aarch64
lapack-devel-3.4.2-8.el7.aarch64
gcc-gfortran-4.8.5-39.el7.aarch64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为只安装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
```

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**，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（3.7.0~ 3.7.11），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，“--enable-loadable-sqlite-extensions”参数用于加载sqlite-devel依赖。

   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.8 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.6 pip3 install numpy报错**解决。
- 如果安装scipy报错，请参考**B.5 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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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，请以实际情况为准）。

```
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，则如下命令修改为只安装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**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（3.7.0~ 3.7.11），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.8 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**章节获取两种架构的开发套件包Ascend-cann-toolkit_*{version}*_linux-*{arch}*.run。
- 安装开发套件包前请确保安装目录可用空间大于7G，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**

说明

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

- 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。

安装完成后，若显示如下信息，则说明软件安装成功：

```
[INFO] xxx install success
```

*xxx*表示安装的实际软件包名。

----结束

6.4.2 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。

6.4.3 配置交叉编译环境

针对Atlas 200 AI加速模块（RC场景）和Atlas 500 智能小站，若使用一台X86服务器（或者PC）搭建开发环境，需要在开发环境安装交叉编译工具，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	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，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）： **sudo apt-get install g++-aarch64-linux-gnu**

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_*{version}*_linux-*{arch}*.run。
- 安装开发套件包前请确保安装目录可用空间大于7G，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。
- 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。

安装完成后，若显示如下信息，则说明软件安装成功：

```
[INFO] xxx install success
```

*xxx*表示安装的实际软件包名。

----结束

说明

涉及动态shape网络的场景下须安装二进制算子包，具体操作请参考**A.1 安装、升级和卸载二进制算子包**。

6.5.2 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：

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。其中软件包安装详细日志路径如下。
   a. root用户：“/var/log/ascend_seclog/ascend_tfplugin_install.log”
   b. 非root用户：“*${HOME}*/var/log/ascend_seclog/ascend_tfplugin_install.log”
   c. 其中${HOME}为当前用户目录。

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_v0.0.4_6.3.RC2.alpha001 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。

6.5.3 安装深度学习框架

6.5.3.1 安装 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。

  a. 编译安装HDF5。

  i. 使用wget下载HDF5源码包，可以下载到安装环境的任意目录，命令为：
  ```
  wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.5.tar.gz --no-check-certificate
  ```
  ii. 进入下载后的目录，解压源码包，命令为：
  ```
  tar -zxvf hdf5-1.10.5.tar.gz
  ```
  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
  ```
  iii. 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
  ```

  b. 安装h5py。

  root用户下安装h5py依赖包命令如下。
  ```
  pip3 install Cython
  ```
  安装h5py命令如下：
  ```
  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",
      ],
  )
```

步骤6 继续从官方的“配置build”（**TensorFlow官网**）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：

```
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
  ```

----结束

6.5.3.2 安装 PyTorch

安装前请参考完成相应配套版本CANN软件的安装。若用户需要详细了解CANN软件和其安装流程，可从**1 安装须知**章节开始了解手册内容。若用户仅进行离线推理，请跳过此章节。

昇腾开发PyTorch Adapter插件用于适配PyTorch框架，为使用PyTorch框架的开发者提供昇腾AI处理器的超强算力，本章节指导用户安装PyTorch框架。

对应分支代码包下载

说明

PyTorch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）。

安装PyTorch时，请参见下载对应分支代码包。

安装 PyTorch 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 wheel --user**。

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

安装 PyTorch 1.8.1 或 1.11.0

推荐用户使用编好的二进制whl包进行安装。

步骤1 安装官方torch包。

- x86_64

  ```
  # 安装1.8.1版本
  pip3 install torch==1.8.1+cpu
  # 安装1.11.0版本
  pip3 install torch==1.11.0+cpu
  ```

  若执行以上命令安装cpu版本PyTorch报错，请点击下方PyTorch官方链接下载whl包安装。

  PyTorch 1.8.1版本：**下载链接**。

  PyTorch 1.11.0版本：**下载链接**。

- aarch64

  a. 进入安装目录，执行如下命令获取鲲鹏文件共享中心上对应版本的whl包。

  ```
  # 安装1.8.1版本
  wget https://repo.huaweicloud.com/kunpeng/archive/Ascend/PyTorch/torch-1.8.1-cp37-cp37m-linux_aarch64.whl
  # 安装1.11.0版本
  wget https://repo.huaweicloud.com/kunpeng/archive/Ascend/PyTorch/torch-1.11.0-cp37-cp37m-linux_aarch64.whl
  ```

  b. 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**。

  ```
  # 安装1.8.1版本
  pip3 install torch-1.8.0a0+56b43f4-cp37-cp37m-linux_aarch64.whl
  # 安装1.11.0版本
  pip3 install torch-1.11.0a0+gitbc2c6ed-cp37-cp37m-linux_aarch64.whl
  ```

步骤2 安装PyTorch插件torch_npu。以下命令以在aarch64架构下安装为例。

1. 进入安装目录，执行如下命令获取PyTorch插件的whl包。
```
# 若用户在x86架构下安装插件，请将命令中文件包名中的“aarch64”改为“x86_64”。
# 安装1.8.1版本
wget https://gitee.com/ascend/pytorch/releases/download/v3.0.0-pytorch1.8.1/torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 安装1.11.0版本
wget https://gitee.com/ascend/pytorch/releases/download/v3.0.0-pytorch1.11.0/torch_npu-1.11.0rc2-cp37-cp37m-linux_aarch64.whl
```
2. 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**。
```
# 若用户在x86架构下安装插件，请将命令中文件包名中的“aarch64”改为“x86_64”。
# 安装1.8.1版本
pip3 install torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 安装1.11.0版本
pip3 install torch_npu-1.11.0rc2-cp37-cp37m-linux_aarch64.whl
```

步骤3 安装对应框架版本的torchvision。
```
#PyTorch 1.8.1需安装0.9.1版本，PyTorch 1.11.0需安装0.12.0版本
pip3 install torchvision==0.9.1
```

----结束

编译安装前准备

在安装官方torch包和PyTorch插件时，用户也可选择编译安装方式安装，此时需要安装系统依赖。

- 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 #gcc7.3.0版本及以上，cmake3.12.0版本及以上。若用户要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。
    ```
  - 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。若用户要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。
    ```

编译安装 PyTorch 1.8.1 或 1.11.0

以下操作步骤以安装PyTorch 1.8.1版本为例。

步骤1 安装官方torch包。

- x86_64
  ```
  pip3 install torch==1.8.1+cpu
  ```
  若执行以上命令安装cpu版本PyTorch报错，请点击下方PyTorch官方链接下载whl包安装。

  PyTorch 1.8.1版本：**下载链接**。

  PyTorch 1.11.0版本：**下载链接**。
- 在aarch64架构下，用户可以选择编译安装官方torch包。
  a. 下载PyTorch v1.8.1源码包，1.11.0版本请替换版本号为v1.11.0。
  ```
  git clone -b v1.8.1 https://github.com/pytorch/pytorch.git --depth=1 pytorch_v1.8.1
  ```
  b. 进入源码包获取被动依赖代码。
  ```
  cd pytorch_v1.8.1
  git submodule sync
  git submodule update --init --recursive
  ```
  c. 配置环境变量。
  ```
  export USE_XNNPACK=0
  ```


d. 执行编译安装。
```
python3 setup.py install
```

步骤2 编译生成PyTorch插件的二进制安装包。
```
# 下载对应PyTorch版本分支代码，进入插件根目录
git clone -b v5.0.rc1.alpha003-pytorch1.8.1 https://gitee.com/ascend/pytorch.git
cd pytorch
# 指定Python版本编包方式，以Python3.7为例，其他Python版本请使用 --python=3.8或--python3.9
bash ci/build.sh --python=3.7
```

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**。
```
pip3 install --upgrade dist/torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 若用户在x86架构下安装插件，或安装1.11.0版本，请替换为对应的whl包。
```

步骤4 安装对应框架版本的torchvision。
```
#PyTorch 1.8.1需安装0.9.1版本，PyTorch 1.11.0需安装0.12.0版本
pip3 install torchvision==0.9.1
```

步骤5 配置环境变量，验证是否安装成功。

1. 请参考**配置环境变量**章节配置CANN环境变量脚本。
2. 执行单元测试脚本，验证PyTorch是否安装成功。
```
cd test/test_network_ops/
python3 test_div.py
```
结果显示OK证明PyTorch框架与插件安装成功。

----结束

安装 APEX 混合精度模块

- 在PyTorch 1.8.1和1.11.0版本下，推荐用户使用编好的二进制whl包安装APEX模块。
  a. 进入安装目录，执行如下命令获取获取PyTorch版本对应的APEX模块whl包。
```
# 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”。
# 1.8.1版本
wget https://gitee.com/ascend/apex/releases/download/v3.0.0-1.8.1/apex-0.1_ascend-cp37-cp37m-linux_aarch64.whl
# 1.11.0版本
wget https://gitee.com/ascend/apex/releases/download/v3.0.0-1.11.0/apex-0.1_ascend-cp37-cp37m-linux_aarch64.whl
```
  b. 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**。
```
# 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
```
- PyTorch 1.5.0版本配套APEX模块只支持编译安装。使用编译安装APEX模块请参考相关**README文档**。

混合精度训练可以提升模型的性能，用户可以根据场景选择引入APEX混合精度模块或使用AscendPyTorch 1.8.1及以上版本集成了的AMP模块。APEX模块有4种功能模式可选择，可以提供不同场景下的混合精度训练支持。AMP功能只类似APEX模块中的一种功能，但无需引入可以直接使用。AMP与APEX模块的使用介绍可参考《**PyTorch 网络模型迁移和训练指南**》中的“混合精度说明“章节。

6.5.3.3 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。

6.5.4 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服务器任一目录并进行解压（**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

6.5.5 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。本章节只介绍使用HCCN Tool工具配置网络的命令，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《**Ascend 910 6.0.0 HCCN Tool 接口参考（AI加速卡）**》。

Atlas 800 训练服务器、Atlas 900 AI 集群场景

说明

判定是SMP模式还是AMP模式，请登录BMC后台执行命令“ipmcget -d npuworkmode”进行查询。

- SMP（对称多处理器）模式：

  以root用户登录到AI Server配置每个device的网卡IP。配置要求：

  - AI Server中的第0/4，1/5，2/6，3/7号网卡需处于同一网段，第0/1/2/3号网卡在不同网段，第4/5/6/7号网卡在不同网段。
  - 对于集群场景，各AI Server对应的位置的device需处于同一网段，例如AI Server1和AI Server2的0号网卡需处于同一网段，AI Server1和AI Server2的1号网卡需处于同一网段。IP地址需要根据实际情况修改。

```
hccn_tool -i 0 -ip -s address 192.168.100.101 netmask 255.255.255.0
hccn_tool -i 1 -ip -s address 192.168.101.101 netmask 255.255.255.0
hccn_tool -i 2 -ip -s address 192.168.102.101 netmask 255.255.255.0
hccn_tool -i 3 -ip -s address 192.168.103.101 netmask 255.255.255.0
hccn_tool -i 4 -ip -s address 192.168.100.100 netmask 255.255.255.0
hccn_tool -i 5 -ip -s address 192.168.101.100 netmask 255.255.255.0
hccn_tool -i 6 -ip -s address 192.168.102.100 netmask 255.255.255.0
hccn_tool -i 7 -ip -s address 192.168.103.100 netmask 255.255.255.0
```

- AMP（非对称多处理器）模式：

  AMP模式下暂不需要配置device的网卡IP。

说明

6.5.6 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
. /usr/local/Ascend/ascend-toolkit/set_env.sh

# 安装tfplugin包时配置
. /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：

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。

7 安装运行环境（nnrt 软件，在物理机安装）

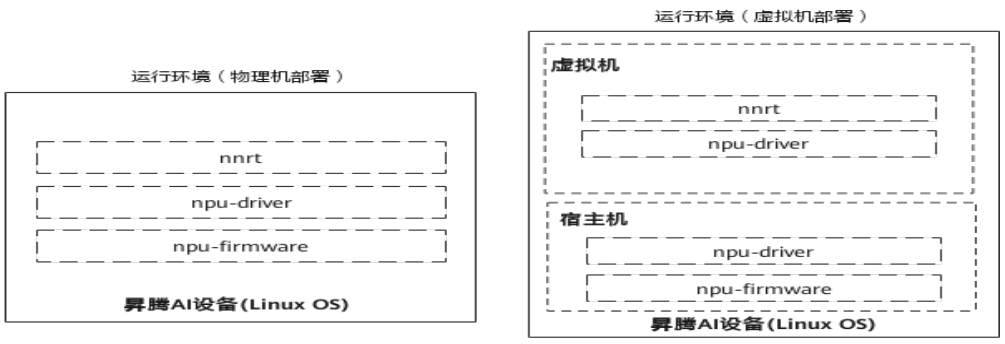
7.1 安装前必读

参考本章节安装运行环境，仅支持离线推理。

运行环境是实际运行用户开发的应用程序的环境。

- 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）》**。
- Atlas 500 智能小站出厂预安装Euler OS、驱动&固件，Atlas 500 智能小站的初始配置请参考**《Atlas 500 智能小站 用户指南（型号 3000）》**。
- 其他昇腾设备运行环境（仅支持离线推理）安装软件如**图7-1**所示。用户可进行物理机部署。

图 7-1 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。

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 》，对从Support网站下载的软件包进行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.5 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，详情请参见**A.7 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 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

- 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。
- 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。

安装完成后，若显示如下信息，则说明软件安装成功：

```
[INFO] xxx install success
```

*xxx*表示安装的实际软件包名。

----结束

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。

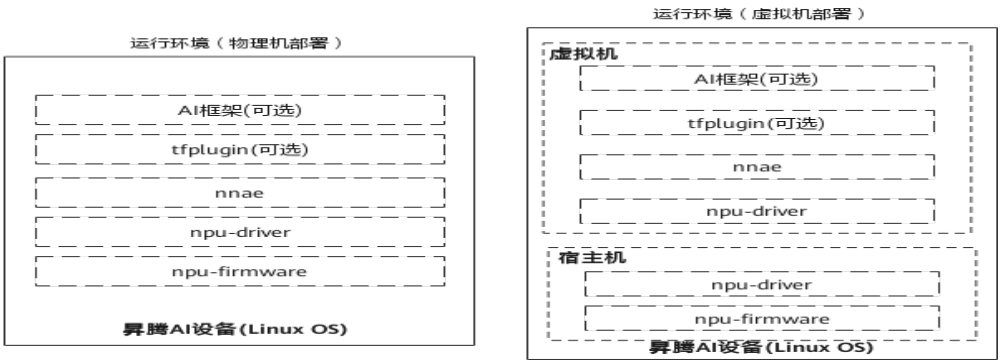
8 安装运行环境（ nnae 软件，在物理机安装 ）

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。

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
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。

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 》，对从Support网站下载的软件包进行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。

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.5 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，详情请参见**A.7 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 18.04、CentOS 7.6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、Linx系统可参考Ubuntu 18.04进行安装。
- EulerOS、OpenEuler、CentOS、BCLinux、Kylin、UOS20 1020e系统可参考CentOS 7.6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 18.04

表 8-2 依赖信息

类别	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）。
cmake	>=3.5.1
make	-
gcc g++	• 离线推理场景 要求4.8.5版本及以上gcc。 • 在线推理、训练、Ascend Graph开发场景 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。若使用PyTorch 1.11.0，要求安装7.5.0及以上版本gcc。
zlib1g zlib1g-dev libsqlite3-dev openssl libssl-dev libffi-dev unzip pciutils net-tools libblas-dev gfortran libblas3 liblapack-dev liblapack3	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

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

CentOS 7.6

表 8-3 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）。
cmake	>=3.5.1
make	-
gcc gcc-c++	• 离线推理场景 要求4.8.5版本及以上gcc。 • 在线推理、训练、Ascend Graph开发场景 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。若使用PyTorch 1.11.0，要求安装7.5.0及以上版本gcc。
unzip zlib-devel libffi-devel openssl-devel pciutils net-tools sqlite-devel lapack-devel gcc-gfortran	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

类别	版本限制
protobuf	>=3.11.3
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）。
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

类别	版本限制
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 18.04）

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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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 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，请以实际情况为准）。

```
gcc (Ubuntu 7.3.0-3ubuntu1~18.04) 7.3.0
g++ (Ubuntu 7.3.0-3ubuntu1~18.04) 7.3.0
GNU Make 4.1
cmake version 3.10.2
zlib1g:arm64   1:1.2.11.dfsg-0ubuntu2 arm64       compression library - runtime
zlib1g-dev:arm64 1:1.2.11.dfsg-0ubuntu2 arm64       compression library - development
libsqlite3-dev:arm64 3.22.0-1ubuntu0.3 arm64       SQLite 3 development files
openssl  1.1.1-1ubuntu2.1~18.04.6 arm64   Secure Sockets Layer toolkit - cryptographic utility
libssl-dev:arm64 1.1.1-1ubuntu2.1~18.04.6 arm64     Secure Sockets Layer toolkit - development files
libffi-dev:arm64 3.2.1-8      arm64       Foreign Function Interface library (development files)
unzip          6.0-21ubuntu1 arm64       De-archiver for .zip files
pciutils       1:3.5.2-1ubuntu1 arm64       Linux PCI Utilities
net-tools      1.60+git20161116.90da8a0-1ubuntu1 arm64       NET-3 networking toolkit
libblas-dev:arm64 3.7.1-4ubuntu1 arm64       Basic Linear Algebra Subroutines 3, static library
gfortran       4:7.4.0-1ubuntu2.3 arm64       GNU Fortran 95 compiler
libblas3:arm64 3.7.1-4ubuntu1 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，则如下命令修改为只安装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
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**，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（ 3.7.0~ 3.7.11 ），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，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libsqlite3-dev依赖。

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.8 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.7 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（ CentOS 7.6 ）

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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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，修改线程最大数，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
```

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，请以实际情况为准）。

```
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39)
g++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39)
GNU Make 3.82
cmake version 2.8.12.2
unzip-6.0-21.el7.aarch64
zlib-devel-1.2.7-18.el7.aarch64
libffi-devel-3.0.13-18.el7.aarch64
openssl-devel-1.0.2k-19.el7.aarch64
pciutils-3.5.1-3.el7.aarch64
net-tools-2.0-0.25.20131004git.el7.aarch64
sqlite-devel-3.7.17-8.el7_7.1.aarch64
lapack-devel-3.4.2-8.el7.aarch64
gcc-gfortran-4.8.5-39.el7.aarch64
```

否则请执行如下安装命令（如果只有部分软件未安装，则如下命令修改为只安装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
```

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**，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（ 3.7.0~ 3.7.11 ），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，“--enable-loadable-sqlite-extensions”参数用于加载sqlite-devel依赖。

   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.8 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.6 pip3 install numpy报错**解决。
- 如果安装scipy报错，请参考**B.5 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。如需配置网络代理，请参见**A.6 配置系统网络代理**。

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，请以实际情况为准）。

```
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，则如下命令修改为只安装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（3.7.0~ 3.7.11），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.8 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昇腾芯片驱动。
- 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 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

- 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。
- 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。

安装完成后，若显示如下信息，则说明软件安装成功：

```
[INFO] xxx install success
```

*xxx*表示安装的实际软件包名。

----结束

说明

涉及动态shape网络的场景下须安装二进制算子包，具体操作请参考**A.1 安装、升级和卸载二进制算子包**。

8.6 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：

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。其中软件包安装详细日志路径如下。
   a. root用户：“/var/log/ascend_seclog/ascend_tfplugin_install.log”
   b. 非root用户：“*${HOME}*/var/log/ascend_seclog/ascend_tfplugin_install.log”
   c. 其中${HOME}为当前用户目录。

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_v0.0.4_6.3.RC2.alpha001 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。

8.7 安装深度学习框架

8.7.1 安装 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。

  a. 编译安装HDF5。

     i. 使用wget下载HDF5源码包，可以下载到安装环境的任意目录，命令为：

     ```
     wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.5.tar.gz --no-check-certificate
     ```

     ii. 进入下载后的目录，解压源码包，命令为：

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

     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
     ```

     iii. 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
     ```

  b. 安装h5py。

     root用户下安装h5py依赖包命令如下。

     ```
     pip3 install Cython
     ```

     安装h5py命令如下：

     ```
     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",
      ],
  )
```

步骤6 继续从官方的“配置build”（**TensorFlow官网**）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：

```
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
  ```

----结束

8.7.2 安装 PyTorch

安装前请参考完成相应配套版本CANN软件的安装。若用户需要详细了解CANN软件和其安装流程，可从**1 安装须知**章节开始了解手册内容。若用户仅进行离线推理，请跳过此章节。

昇腾开发PyTorch Adapter插件用于适配PyTorch框架，为使用PyTorch框架的开发者提供昇腾AI处理器的超强算力，本章节指导用户安装PyTorch框架。

对应分支代码包下载

说明

PyTorch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）。

安装PyTorch时，请参见下载对应分支代码包。

安装 PyTorch 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 wheel --user**。

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

安装 PyTorch 1.8.1 或 1.11.0

推荐用户使用编好的二进制whl包进行安装。

步骤1 安装官方torch包。

- x86_64

  ```
  # 安装1.8.1版本
  pip3 install torch==1.8.1+cpu
  # 安装1.11.0版本
  pip3 install torch==1.11.0+cpu
  ```

  若执行以上命令安装cpu版本PyTorch报错，请点击下方PyTorch官方链接下载whl包安装。

  PyTorch 1.8.1版本：**下载链接**。

  PyTorch 1.11.0版本：**下载链接**。

- aarch64

  a. 进入安装目录，执行如下命令获取鲲鹏文件共享中心上对应版本的whl包。

  ```
  # 安装1.8.1版本
  wget https://repo.huaweicloud.com/kunpeng/archive/Ascend/PyTorch/torch-1.8.1-cp37-cp37m-linux_aarch64.whl
  # 安装1.11.0版本
  wget https://repo.huaweicloud.com/kunpeng/archive/Ascend/PyTorch/torch-1.11.0-cp37-cp37m-linux_aarch64.whl
  ```

  b. 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**。

  ```
  # 安装1.8.1版本
  pip3 install torch-1.8.0a0+56b43f4-cp37-cp37m-linux_aarch64.whl
  # 安装1.11.0版本
  pip3 install torch-1.11.0a0+gitbc2c6ed-cp37-cp37m-linux_aarch64.whl
  ```

步骤2 安装PyTorch插件torch_npu。以下命令以在aarch64架构下安装为例。

1. 进入安装目录，执行如下命令获取PyTorch插件的whl包。


```
# 若用户在x86架构下安装插件，请将命令中文件包名中的“aarch64”改为“x86_64”。
# 安装1.8.1版本
wget https://gitee.com/ascend/pytorch/releases/download/v3.0.0-pytorch1.8.1/torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 安装1.11.0版本
wget https://gitee.com/ascend/pytorch/releases/download/v3.0.0-pytorch1.11.0/torch_npu-1.11.0rc2-cp37-cp37m-linux_aarch64.whl
```

2. 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**。

```
# 若用户在x86架构下安装插件，请将命令中文件包名中的“aarch64”改为“x86_64”。
# 安装1.8.1版本
pip3 install torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 安装1.11.0版本
pip3 install torch_npu-1.11.0rc2-cp37-cp37m-linux_aarch64.whl
```

步骤3 安装对应框架版本的torchvision。

```
#PyTorch 1.8.1需安装0.9.1版本，PyTorch 1.11.0需安装0.12.0版本
pip3 install torchvision==0.9.1
```

----结束

编译安装前准备

在安装官方torch包和PyTorch插件时，用户也可选择编译安装方式安装，此时需要安装系统依赖。

- 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 #gcc7.3.0版本及以上，cmake3.12.0版本及以上。若用户要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。
    ```

  - 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。若用户要安装1.11.0版本PyTorch，则gcc需为7.5.0版本以上。
    ```

编译安装 PyTorch 1.8.1 或 1.11.0

以下操作步骤以安装PyTorch 1.8.1版本为例。

步骤1 安装官方torch包。

- x86_64

  ```
  pip3 install torch==1.8.1+cpu
  ```

  若执行以上命令安装cpu版本PyTorch报错，请点击下方PyTorch官方链接下载whl包安装。

  PyTorch 1.8.1版本：**下载链接**。

  PyTorch 1.11.0版本：**下载链接**。

- 在aarch64架构下，用户可以选择编译安装官方torch包。

  a. 下载PyTorch v1.8.1源码包，1.11.0版本请替换版本号为v1.11.0。

  ```
  git clone -b v1.8.1 https://github.com/pytorch/pytorch.git --depth=1 pytorch_v1.8.1
  ```

  b. 进入源码包获取被动依赖代码。

  ```
  cd pytorch_v1.8.1
  git submodule sync
  git submodule update --init --recursive
  ```

  c. 配置环境变量。

  ```
  export USE_XNNPACK=0
  ```


d. 执行编译安装。
```
python3 setup.py install
```

步骤2 编译生成PyTorch插件的二进制安装包。
```
# 下载对应PyTorch版本分支代码，进入插件根目录
git clone -b v5.0.rc1.alpha003-pytorch1.8.1 https://gitee.com/ascend/pytorch.git
cd pytorch
# 指定Python版本编包方式，以Python3.7为例，其他Python版本请使用 --python=3.8或--python3.9
bash ci/build.sh --python=3.7
```

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**。
```
pip3 install --upgrade dist/torch_npu-1.8.1-cp37-cp37m-linux_aarch64.whl
# 若用户在x86架构下安装插件，或安装1.11.0版本，请替换为对应的whl包。
```

步骤4 安装对应框架版本的torchvision。
```
#PyTorch 1.8.1需安装0.9.1版本，PyTorch 1.11.0需安装0.12.0版本
pip3 install torchvision==0.9.1
```

步骤5 配置环境变量，验证是否安装成功。

1. 请参考**配置环境变量**章节配置CANN环境变量脚本。
2. 执行单元测试脚本，验证PyTorch是否安装成功。
   ```
   cd test/test_network_ops/
   python3 test_div.py
   ```
   结果显示OK证明PyTorch框架与插件安装成功。

----结束

安装 APEX 混合精度模块

- 在PyTorch 1.8.1和1.11.0版本下，推荐用户使用编好的二进制whl包安装APEX模块。
  a. 进入安装目录，执行如下命令获取获取PyTorch版本对应的APEX模块whl包。
  ```
  # 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”。
  # 1.8.1版本
  wget https://gitee.com/ascend/apex/releases/download/v3.0.0-1.8.1/apex-0.1_ascend-cp37-cp37m-linux_aarch64.whl
  # 1.11.0版本
  wget https://gitee.com/ascend/apex/releases/download/v3.0.0-1.11.0/apex-0.1_ascend-cp37-cp37m-linux_aarch64.whl
  ```
  b. 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**。
  ```
  # 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
  ```
- PyTorch 1.5.0版本配套APEX模块只支持编译安装。使用编译安装APEX模块请参考相关**README文档**。

混合精度训练可以提升模型的性能，用户可以根据场景选择引入APEX混合精度模块或使用AscendPyTorch 1.8.1及以上版本集成了的AMP模块。APEX模块有4种功能模式可选择，可以提供不同场景下的混合精度训练支持。AMP功能只类似APEX模块中的一种功能，但无需引入可以直接使用。AMP与APEX模块的使用介绍可参考《**PyTorch 网络模型迁移和训练指南**》中的“混合精度说明“章节。

8.7.3 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。

8.8 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服务器任一目录并进行解压（**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

8.9 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

# 安装tfplugin包时配置
. /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：

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。

8.10 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。本章节只介绍使用HCCN Tool工具配置网络的命令，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《**Ascend 910 6.0.0 HCCN Tool 接口参考（AI加速卡）**》。

Atlas 800 训练服务器、Atlas 900 AI 集群场景

说明

判定是SMP模式还是AMP模式，请登录BMC后台执行命令“ipmcget -d npuworkmode”进行查询。

- SMP（对称多处理器）模式：

  以root用户登录到AI Server配置每个device的网卡IP。配置要求：

  - AI Server中的第0/4，1/5，2/6，3/7号网卡需处于同一网段，第0/1/2/3号网卡在不同网段，第4/5/6/7号网卡在不同网段。
  - 对于集群场景，各AI Server对应的位置的device需处于同一网段，例如AI Server1和AI Server2的0号网卡需处于同一网段，AI Server1和AI Server2的1号网卡需处于同一网段。IP地址需要根据实际情况修改。

```
hccn_tool -i 0 -ip -s address 192.168.100.101 netmask 255.255.255.0
hccn_tool -i 1 -ip -s address 192.168.101.101 netmask 255.255.255.0
hccn_tool -i 2 -ip -s address 192.168.102.101 netmask 255.255.255.0
hccn_tool -i 3 -ip -s address 192.168.103.101 netmask 255.255.255.0
hccn_tool -i 4 -ip -s address 192.168.100.100 netmask 255.255.255.0
hccn_tool -i 5 -ip -s address 192.168.101.100 netmask 255.255.255.0
hccn_tool -i 6 -ip -s address 192.168.102.100 netmask 255.255.255.0
hccn_tool -i 7 -ip -s address 192.168.103.100 netmask 255.255.255.0
```

- AMP（非对称多处理器）模式：

  AMP模式下暂不需要配置device的网卡IP。

说明

9 升级

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）。

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。

表 9-2 CANN 软件包升级路径说明

原版本	当前版本	是否支持升级
CANN 3.1.0	CANN 6.0.0	否。请先执行卸载，再进行安装操作。
CANN 3.2.0	CANN 6.0.0	否。请先执行卸载，再进行安装操作。
CANN 3.3.0	CANN 6.0.0	否。请先执行卸载，再进行安装操作。
CANN 5.0.2	CANN 6.0.0	否。请先执行卸载，再进行安装操作。
CANN 5.0.3	CANN 6.0.0	是。
CANN 5.0.4	CANN 6.0.0	是。
CANN 5.0.5	CANN 6.0.0	是。
CANN 5.1.RC1	CANN 6.0.0	是。
CANN 6.0.RC1	CANN 6.0.0	是。

9.2 升级操作

升级流程

- 可单独升级CANN软件。
- 如果要升级固件和驱动，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9-3参考对应文档进行升级操作。

表 9-3 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 SoC A1	请参见《Atlas 200I SoC A1核心板 6.0.0 NPU驱动和固件升级指南》。

产品型号	参考文档
A300-3000	请参见《Atlas 300I 推理卡 6.0.0 NPU驱动和固件升级指南（型号 3000, 3010）》。
A300-3010	请参见《Atlas 300I 推理卡 6.0.0 NPU驱动和固件升级指南（型号 3000, 3010）》。
Atlas 300I Pro	请参见《Ascend 310P 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
Atlas 300V Pro	请参见《Ascend 310P 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
Atlas 300V	请参见《Ascend 310P 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
Atlas 300I Duo	请参见《Ascend 310P 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
A300T-9000	请参见《Ascend 910 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
Atlas 300T Pro	请参见《Ascend 910 6.0.0 NPU驱动和固件升级指南（AI加速卡）》。
A500 Pro-3000	请参见《Atlas 300I 推理卡 6.0.0 NPU驱动和固件升级指南（型号 3000, 3010）》。
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《Ascend 910 6.0.0 NPU驱动和固件升级指南（AI服务器）》。
A800-9010	请参见《Ascend 910 6.0.0 NPU驱动和固件升级指南（AI服务器）》。
A900-9000	请参见《Ascend 910 6.0.0 NPU驱动和固件升级指南（AI服务器）》。

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7G，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 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 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
  ```

----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。

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。

----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**》。

A 常用操作

A.1 安装、升级和卸载二进制算子包

涉及动态shape网络的场景下须安装二进制算子包，二进制算子包支持安装、升级和卸载功能。

约束

二进制算子包依赖软件包nnae或toolkit，执行安装和升级操作时，当前环境需已安装配套版本的nnae或toolkit，并使用同一用户安装。

下载软件包

软件安装前，请参考**表A-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。

表 A-1 CANN 软件包

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	PyTorch训练场景下涉及动态shape时使用。	**获取链接**

说明

*{chip_type}*表示芯片类型，*{version}*表示软件版本号。

二进制算子包相关操作（适用于.run 格式）

二进制算子包支持安装、升级和卸载等操作，用户根据实际需要选择相应参数，可参见**表A-2**。

以安装二进制算子包为例，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
```

若环境中同时安装了配套版本的nnae和toolkit软件包，安装或升级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*>**，来识别到具体的安装目录。
- 方式二：先配置需依赖的nnae或toolkit的环境变量，再执行安装或升级。示例如下（以root用户默认安装路径为例）：

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

二进制算子包可通过.run格式的软件包单独卸载，且支持通过软件包nnae或toolkit统一卸载，具体操作请参考**10 卸载**。

表 A-2 算子包支持的参数说明

参数	说明
--help | -h	查询帮助信息。
--version	查询版本信息。
--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	不使用x11模式运行。
--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。
--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。
--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。
--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。

参数	说明
--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，当环境上存在全局配置文件“ascend_cann_install.info”时，支持使用该参数，但指定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，需先卸载原路径下的CANN软件包并确保全局配置文件“ascend_cann_install.info”已被删除。 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，运行用户需要对指定的安装路径有可读写权限。
--uninstall	卸载已安装的软件。
--upgrade	升级已安装的软件。
--type=<package_type>	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”一起使用。

A.2 安装、回退或卸载 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-3**。

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-3 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。

参数	说明
--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.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
```

> **注意**
>
> 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 操作系统配置网卡 IP 地址

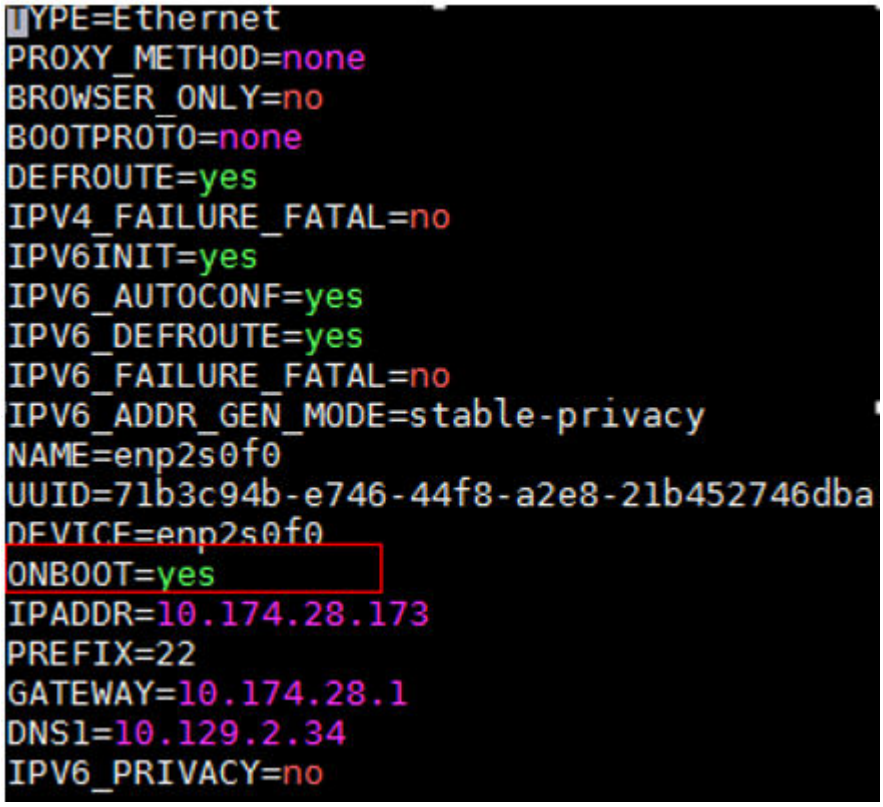
步骤1 以root用户进入OS界面。

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



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 配置系统网络代理

以下步骤是配置网络代理的通用方法，不一定能适配所有的网络环境，网络代理的配置方式以现场实际的网络环境为准。

前提条件

- 确保服务器网线已连接，代理服务器能连接外网。
- 配置代理是建立在服务器处在内部网络，无法直接连接外部网络的条件下。

配置系统网络代理

步骤1 使用root权限登录用户环境。

步骤2 使用如下命令编辑“/etc/profile”文件：

```
vi /etc/profile
```

在文件最后添加如下内容后保存退出（以下仅为示例，请根据实际情况配置）：

```
export http_proxy="http://proxyserverip:port"
export https_proxy="http://proxyserverip:port"
```

其中proxyserverip为代理服务器的ip地址，port为端口。

步骤3 执行如下命令使配置生效。

```
source /etc/profile
```


步骤4 执行如下命令测试是否连接外网。

```
wget www.baidu.com
```

如果能下载到网页html文件则表明服务器已成功连接外网。

说明

使用代理连接网络，如果出现证书错误，则需要先安装代理服务器的证书，才能正常下载第三方组件。

----结束

A.7 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-4。

表 A-4 设置用户有效期

参数	参数说明
-m	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	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。
-d	上一次更改的日期。
-I	停滞时期。过期指定天数后，设定密码为失效状态。
-E	用户到期的日期。超过该日期，此用户将不可用。
-W	用户口令到期前，提前收到警告信息的天数。
-l	列出当前的设置。由非特权用户来确定口令或帐户何时过期。

说明

- 表A-4只列举出常用的参数，用户可通过**chage --help**命令查询详细的参数说明。
- 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.8 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.9 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 run 包安装时提示软件已经安装

问题描述

软件包安装时提示：

```
run package is already installed, install failed
```

解决方案

软件包不支持重复安装，需要先卸载后，再安装。卸载方法请参见**10 卸载**。

B.2 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。

解决方法

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

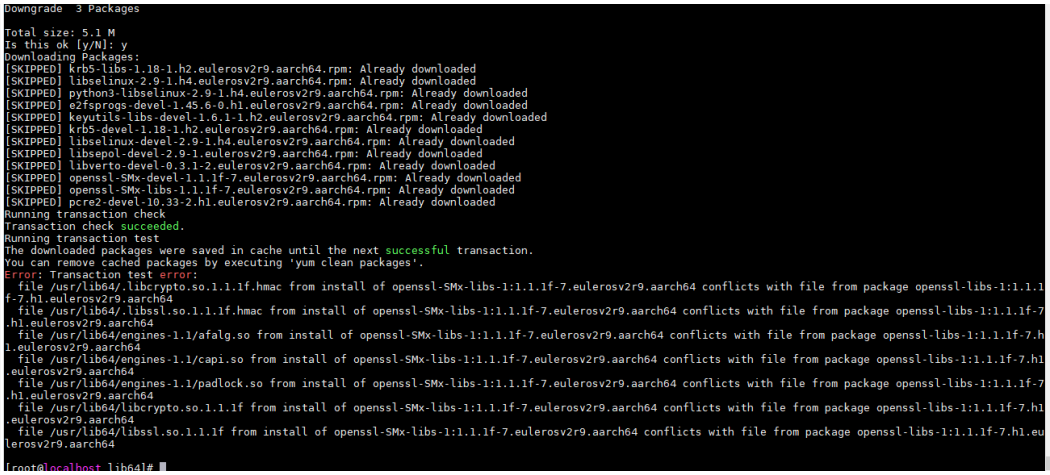
步骤3 执行./uninstall.sh命令运行脚本，完成卸载。

----结束

B.3 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：



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失效。

解决办法

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 --downloaddir=. openssl-devel
```

步骤3 rpm命令强制安装。

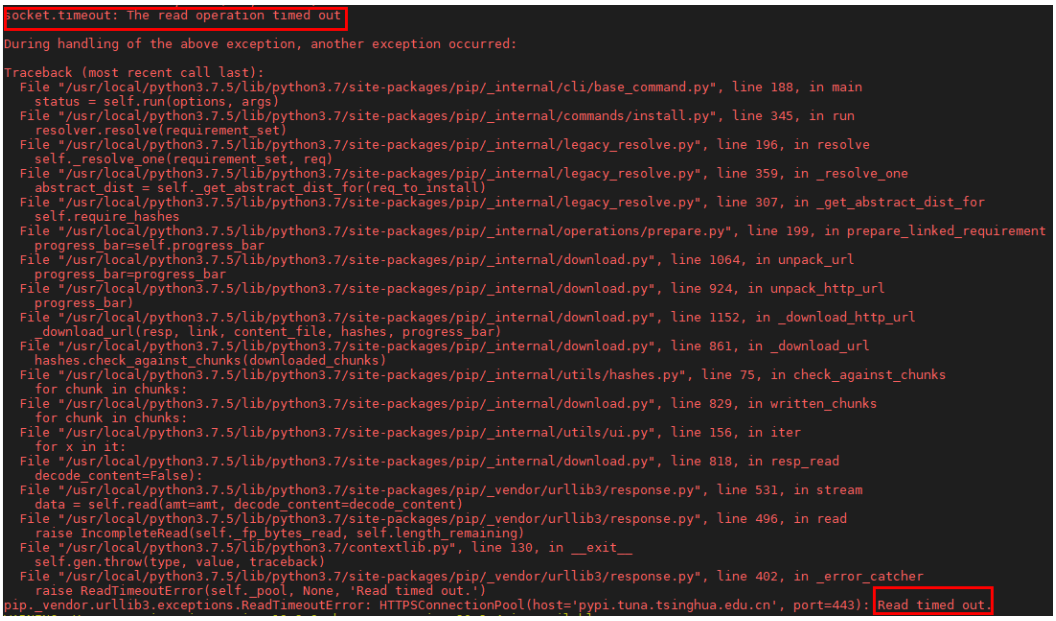
```
rpm -Uvh *.rpm --force
```

----结束

B.4 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：



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。

解决方法

用 - i 参数来指定常用的其它源，如下命令所示：

```
pip3 install scipy -i xxxxxx
```

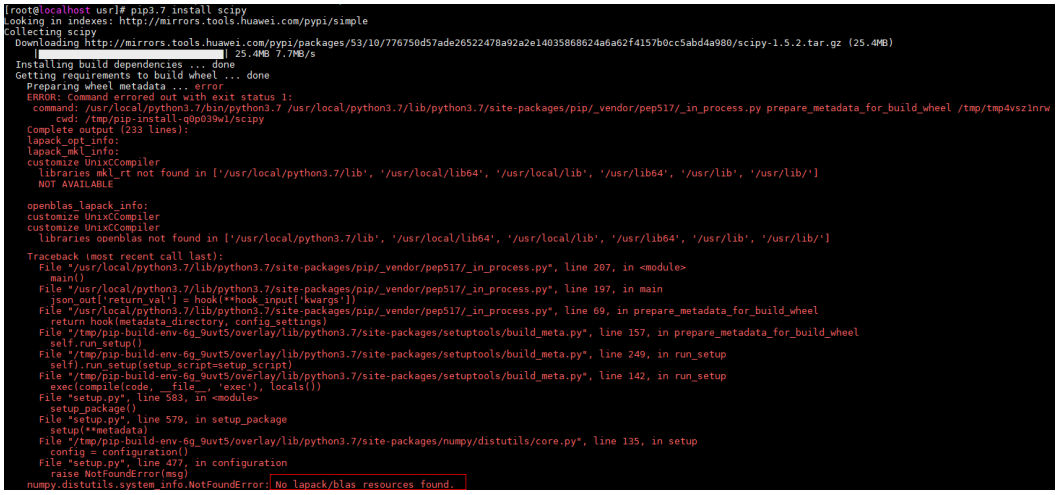
xxxxxx 为客户根据需求自行填写可用的源。

B.5 pip3 install scipy 报错

问题描述

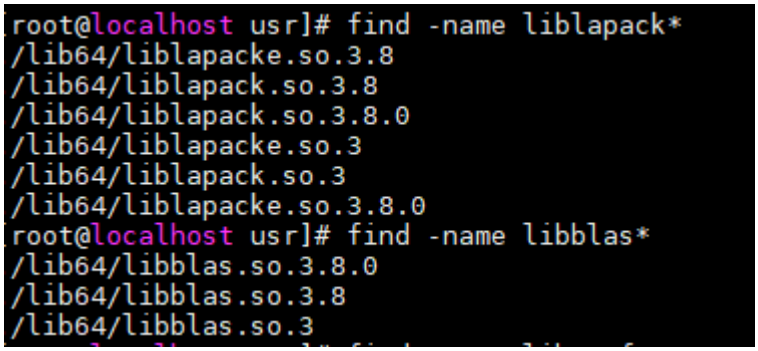
安装scipy时，提示如下错误信息。

图 B-1 错误信息



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。



解决方法

创建liblapack.so.3和libblas.so.3对应的so的软连接。

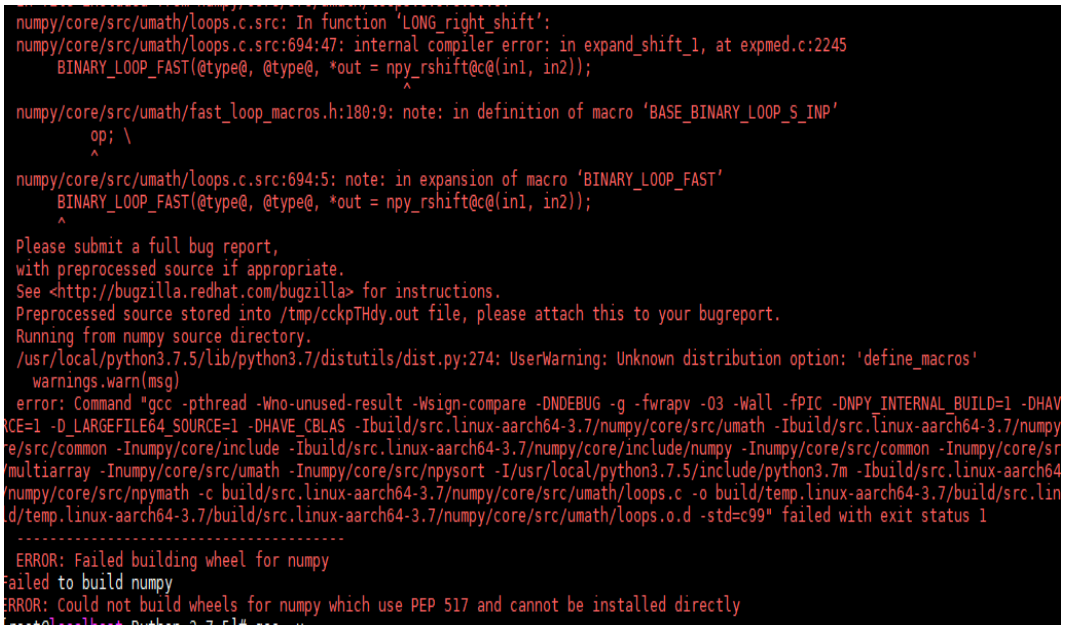
使用软件包的安装用户，执行如下命令：

```
cd /usr/lib64
ln -s libblas.so.3 libblas.so
ln -s liblapack.so.3 liblapack.so
```

B.6 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：



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。

解决方法

执行如下命令安装：

```
export CFLAGS=-std=c99
pip3 install numpy==1.17.2
```

B.7 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。

解决方法

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 执行**pip3 install xxx**命令安装相关软件，查看是否解决。

----结束

B.8 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，可能会出现问题。

解决办法

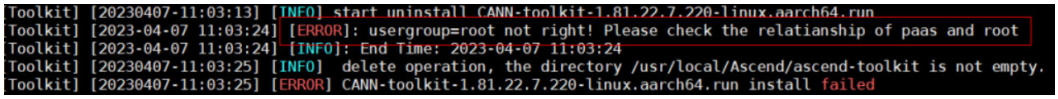
建议使用Python的venv。Python虚拟环境用于将软件包安装与系统隔离开来。

B.9 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



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
```


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
--help | -h	查询帮助信息。
--version	查询版本信息。
--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	不使用x11模式运行。
--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>。

参数	说明
--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，当环境上存在全局配置文件“ascend_cann_install.info”时，支持使用该参数，但指定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，需先卸载原路径下的CANN软件包并确保全局配置文件“ascend_cann_install.info”已被删除。 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，运行用户需要对指定的安装路径有可读写权限。
--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。

参数	说明
--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>	指定芯片型号，以便在安装过程中选择匹配的软件（如AICPU算子包），芯片型号参数chip_type可选范围如下（以软件包toolkit为例）： ● Ascend310，表示芯片型号为Ascend310 PCIe芯片。 ● Ascend310P，表示芯片型号为Ascend310P PCIe芯片。 ● Ascend910，表示芯片型号为Ascend910 PCIe芯片。 ● Ascend310-minirc，表示芯片型号为Ascend310 SoC芯片（与Ascend310相同，但以RC模式启动，作为主控CPU）。 **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。

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：“*${HOME}*/Ascend/nnae/latest/ascend_nnae_install.info”
软件包安装路径的记录路径	root用户：“/etc/Ascend/ascend_cann_install.info” 非root用户：“${HOME}/Ascend/ascend_cann_install.info”

信息说明	路径
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 昇腾产品形态说明

以昇腾AI处理器的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）场景；如果PCIe工作在从模式，则称为EP（Endpoint）场景。

- 昇腾 AI 处理器的工作场景如下：
  - 昇腾310 AI处理器有EP和RC两种场景。
  - 昇腾310P AI处理器只有EP场景。
  - 昇腾910 AI处理器只有EP场景。
- 支持RC场景的产品有：Atlas 200 AI加速模块、Atlas 200 DK开发者套件。

  产品的CPU直接运行用户指定的AI业务软件，接入网络摄像头、I^2C传感器、SPI显示器等其他外挂设备作为从设备接入产品。
- 支持EP场景的产品

  昇腾310 AI处理器：Atlas 200 AI加速模块、Atlas 300I推理卡、Atlas 500智能小站、Atlas 500 Pro智能边缘服务器、Atlas 800推理服务器。

  昇腾310P AI处理器：Atlas 300I Pro推理卡、Atlas 300I Duo推理卡、Atlas 300V视频解析卡、Atlas 300V Pro视频解析卡。

  昇腾910 AI处理器：Atlas 800 训练服务器、Atlas 300T训练卡、Atlas 300T Pro训练卡。

  EP场景通常由Host侧作为主端，Device侧作为从端。客户的AI业务程序运行在Host系统中，产品作为Device系统以PCIe从设备接入Host系统，Host系统通过PCIe通道与Device系统交互，将AI任务加载到Device侧的昇腾 AI 处理器中运行。

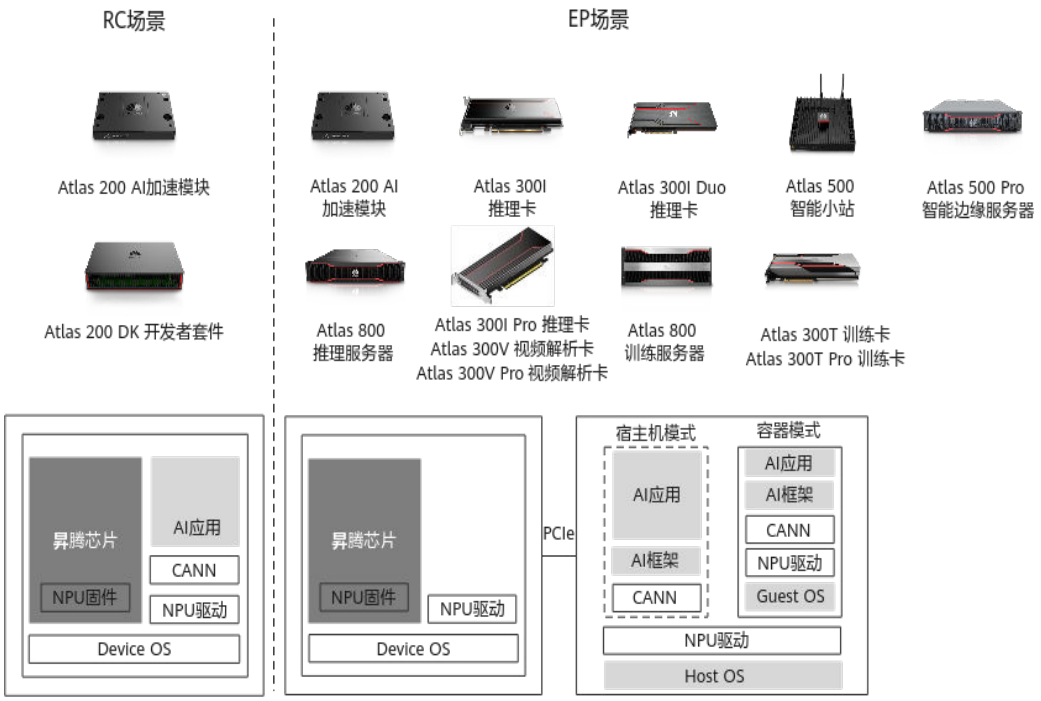
两种场景的产品及架构如图C-1所示。

Host和Device的概念说明如下：

- Host：是指与昇腾AI处理器所在硬件设备相连接的X86服务器、ARM服务器，利用昇腾AI处理器提供的NN（Neural-Network）计算能力完成业务。

- Device：是指安装了昇腾AI处理器的硬件设备，利用PCIe接口与服务器连接，为服务器提供NN计算能力。

图 C-1 RC 和 EP 场景



C.4 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时，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.5 口令复杂度要求

口令至少满足如下要求：

1. 口令长度至少8个字符。
2. 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。

