MindStudio V100R020C20

用户指南

文档版本	01
发布日期	2021-05-19

华为技术有限公司

版权所有 © 华为技术有限公司 2021。 保留一切权利。

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

商标声明

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

注意

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

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 认识 MindStudio.....1
1.1 关于 MindStudio.....1
1.2 What's New.....3
1.3 关键概念.....7
1.4 昇腾产品形态说明.....8
2 开发环境准备.....10
2.1 安装介绍.....10
2.2 安装 MindStudio(物理机、msInstaller 方式）.....11
2.3 安装 MindStudio(物理机、手动方式).....11
2.3.1 安装流程.....12
2.3.2 获取软件包.....12
2.3.3 安装前准备.....13
2.3.3.1 Ubuntu-x86_64 系统.....13
2.3.3.2 Ubuntu-aarch64 系统.....15
2.3.3.3 EulerOS-aarch64 系统.....18
2.3.4 安装 MindStudio.....20
2.3.5 配置编译环境.....26
2.4 安装 MindStudio(容器).....28
2.4.1 安装流程.....28
2.4.2 安装前准备.....28
2.4.3 构建 MindStudio 容器镜像.....30
2.4.3.1 使用 Dockerfile 定制镜像.....30
2.4.4 启动 MindStudio 容器.....32
2.4.4.1 在昇腾 AI 设备上启动 MindStudio 容器.....32
2.4.4.2 在非昇腾 AI 设备上启动 MindStudio 容器.....39
2.4.5 删除 MindStudio 容器.....40
3 算子&模型速查.....41
3.1 查询算子.....41
3.2 查询模型.....42
4 模型转换.....44
4.1 使用前必读.....44
4.2 模型转换.....45

4.3 模型可视化.....63
5 模型训练.....68
5.1 概述.....68
5.2 操作步骤.....69
5.3 MindSpore 模型训练样例参考.....74
5.3.1 准备环境与样例.....75
5.3.2 基于 ResNet50 模板样例的训练步骤.....75
5.3.3 基于 ModelZoo 训练样例的训练步骤.....80
5.4 MindInsight 训练可视化.....85
5.5 MindSpore 模型转换后离线推理.....87
6 脚本转换.....91
6.1 PyTorch GPU2Ascend.....91
6.1.1 概述.....91
6.1.2 生成报告.....91
6.1.3 执行转换.....93
6.2 MindConverter.....94
6.2.1 概述.....94
6.2.2 前提条件.....94
6.2.3 操作步骤.....95
7 应用开发.....101
7.1 准备文档.....101
7.2 基于 MindX SDK 开发应用.....101
7.2.1 创建应用工程.....101
7.2.2 获取与安装 SDK 软件包.....103
7.2.3 开发应用.....105
7.2.3.1 典型业务流程.....105
7.2.3.2 开发流程.....106
7.2.4 可视化流程编排.....107
7.2.5 编译与运行应用工程.....108
7.2.6 SDK 样例工程使用指导.....108
7.3 基于 AscendCL 开发应用.....112
7.3.1 创建应用工程.....112
7.3.2 准备模型文件和数据.....115
7.3.3 添加自定义动态链接库（可选）.....117
7.3.4 开发应用.....118
7.3.5 编译应用工程.....118
7.3.6 运行应用工程.....120
7.4 应用工程调试.....122
7.4.1 调试前准备.....122
7.4.2 断点管理.....124
7.4.3 调试执行.....126

8 自定义算子开发.....133
8.1 使用前必读.....133
8.2 开发流程.....134
8.2.1 TensorFlow TBE 算子开发流程.....134
8.2.2 PyTorch TBE 算子开发流程.....135
8.2.3 MindSpore TBE 算子开发流程.....137
8.2.4 TensorFlow AI CPU 算子开发流程.....138
8.3 算子分析.....139
8.4 工程创建.....142
8.4.1 创建算子工程.....142
8.4.2 设置 Python 库.....146
8.5 TBE 算子开发（TensorFlow）.....151
8.5.1 算子代码实现.....151
8.5.2 算子原型定义.....155
8.5.3 算子信息定义.....157
8.5.4 算子适配插件实现.....161
8.5.5 UT 测试.....161
8.5.5.1 简介.....162
8.5.5.2 接口参考.....162
8.5.5.2.1 OpUT.....162
8.5.5.2.2 BroadcastOpUT.....166
8.5.5.2.3 ElementwiseOpUT.....169
8.5.5.2.4 ReduceOpUT.....171
8.5.5.3 创建和运行 UT 测试用例.....174
8.5.6 算子工程编译.....179
8.5.7 算子部署（本地）.....181
8.5.8 算子部署（远程）.....182
8.5.9 ST 测试.....184
8.6 TBE 算子开发（PyTorch）.....198
8.6.1 算子代码实现.....198
8.6.2 算子原型定义.....203
8.6.3 算子信息定义.....205
8.6.4 UT 测试.....208
8.6.4.1 简介.....208
8.6.4.2 接口参考.....208
8.6.4.2.1 OpUT.....208
8.6.4.2.2 BroadcastOpUT.....213
8.6.4.2.3 ElementwiseOpUT.....215
8.6.4.2.4 ReduceOpUT.....217
8.6.4.3 创建和运行 UT 测试用例.....220
8.6.5 算子工程编译.....226
8.6.6 算子部署（本地）.....227

8.6.7 算子部署（远程）.....228
8.6.8 PyTorch 适配.....230
8.6.9 ST 测试.....230
8.7 TBE 算子开发（MindSpore）.....245
8.7.1 算子实现和注册信息文件.....245
8.7.2 算子原型定义.....245
8.7.3 UT 测试.....246
8.7.3.1 简介.....246
8.7.3.2 接口参考.....246
8.7.3.2.1 OpUT.....246
8.7.3.2.2 BroadcastOpUT.....251
8.7.3.2.3 ElementwiseOpUT.....253
8.7.3.2.4 ReduceOpUT.....255
8.7.3.3 创建和运行 UT 测试用例.....258
8.7.4 ST 测试.....261
8.8 AI CPU 算子开发（TensorFlow）.....267
8.8.1 算子代码实现.....268
8.8.2 算子原型定义.....270
8.8.3 算子信息定义.....272
8.8.4 算子适配插件实现.....273
8.8.5 算子工程编译部署.....274
8.8.6 ST 测试.....274
8.9 其他功能及操作.....274
8.9.1 算子工程管理.....274
8.9.1.1 导入算子工程样例.....274
8.9.1.2 修改算子工程属性.....275
8.9.1.3 切换到算子逻辑视图.....276
8.9.1.4 新增/删除算子.....277
8.9.2 运行生成*.cce 文件.....279
8.9.3 TIK 功能调试.....279
9 精度比对.....281
9.1 简介与约束.....281
9.2 比对数据准备.....283
9.2.1 比对数据格式要求.....283
9.2.2 推理场景数据准备.....284
9.2.2.1 准备离线模型 dump 数据.....284
9.2.2.2 准备 Caffe 模型 npy 数据文件.....287
9.2.2.3 准备 TensorFlow 模型 npy 数据文件.....289
9.2.2.4 准备 ONNX 模型 npy 数据文件.....291
9.2.3 训练场景数据准备.....292
9.2.3.1 准备基于昇腾 AI 处理器运行生成的训练网络 dump 数据.....292
9.2.3.2 准备基于 GPU 运行生成的 TensorFlow 原始训练网络 npy 数据文件.....294

9.3 Vector 比对 296
9.3.1 比对数据说明 296
9.3.2 比对操作 298
9.3.3 子模型导出与比对 303
9.4 FAQ 311
9.4.1 Vector 比对报错：The data in dump file have zero 311
9.4.2 Vector 比对报错：**ModuleNotFoundError: No module named 'google'** 311
9.4.3 如何进行 dump 数据文件与 numpy 文件格式互转 312
9.4.4 如何查看 dump 数据文件 313
9.4.5 如何批量处理生成的 npy 文件名异常情况 314
10 Profiling 性能分析 316
10.1 应用工程 Profiling 316
10.1.1 简介与约束 316
10.1.2 Profiling 前准备 317
10.1.3 启动 Profiling 317
10.1.4 查看 Profiling 数据 319
10.2 导入并展示 Profiling 结果 326
11 System Profiling 工具 328
11.1 功能介绍 328
11.2 使用前准备 328
11.2.1 安装工具依赖 328
11.2.1.1 Ubuntu x86/ARM 328
11.2.1.2 Euler 2.8 ARM 329
11.2.2 使用前准备 330
11.3 使用指南 331
11.3.1 启动 System Profiling 331
11.3.2 Project Explorer 334
11.3.3 TimeLine View 335
11.3.4 Event View 339
11.3.5 Statistics 341
11.3.6 AI Core Metrics 343
11.3.7 Analysis Summary 344
12 AI Core Error 分析工具 348
12.1 功能介绍 348
12.2 总体流程 349
12.3 数据准备 350
12.4 通过工具生成解析数据 353
12.5 问题分析和定位 354
13 操作指南 356
13.1 SSH 连接管理 356
13.1.1 概述 356

13.1.2 新增 SSH 连接 358
13.1.3 删除 SSH 连接 359
13.1.4 修改 SSH 连接 359
13.1.5 复制 SSH 连接 359
13.1.6 加密 SSH 密码 359
13.1.7 修改 SSH 密码保存方式 360
13.2 工程管理 361
13.2.1 打开/关闭/移除 361
13.2.2 新建/删除文件 364
13.3 运行管理 365
13.3.1 修改运行配置 365
13.3.2 中断运行和重新运行 366
13.4 设备管理 366
13.5 ADK 管理 367
13.5.1 概述 367
13.5.2 更新 ADK 369
13.5.3 切换/激活 ADK 369
13.5.4 配置交叉编译环境 371
13.6 日志管理 372
13.6.1 使用前必读 372
13.6.2 日志管理 373
13.6.3 设置日志级别 375
13.7 常用快捷键 376
13.8 常用原生功能 377
14 MindStudio 管理 379
14.1 查询 MindStudio 版本 379
14.2 启动 MindStudio 380
14.3 停止 MindStudio 380
14.4 更新 MindStudio 380
14.5 卸载 MindStudio 380
14.6 FAQ 381
14.6.1 使用 pip3 install 软件时提示" Could not find a version that satisfies the requirement xxx" 381
15 附录 383
15.1 软件包说明 383
15.2 安装依赖 384
15.2.1 安装依赖(Ubuntu-x86_64) 384
15.2.2 安装依赖(Ubuntu-aarch64) 388
15.2.3 安装依赖(EulerOS-aarch64) 393
15.3 算子规格说明 398
15.3.1 Caffe 框架算子规格 398
15.3.2 TensorFlow 框架算子规格 398
15.3.3 ONNX 算子规格 398

15.4 特性配置 398
15.4.1 定制网络修改（Caffe） 399
15.4.1.1 简介 399
15.4.1.2 扩展算子列表 399
15.4.1.3 扩展算子规则 401
15.4.1.4 样例参考 412
15.4.1.4.1 FasterRCNN 网络模型 prototxt 修改 412
15.4.1.4.2 YOLOv3 网络模型 prototxt 修改 414
15.4.1.4.3 YOLOv2 网络模型 prototxt 修改 416
15.4.1.4.4 SSD 网络模型 prototxt 修改 418
15.4.1.4.5 BatchedMatMul 算子 prototxt 修改 419
15.4.1.4.6 SENet 网络模型 prototxt 修改 420
15.5 使用 AIPP 色域转换模型时如何判断视频流的格式标准 421
15.6 配置系统网络代理 421
15.7 安全加固建议 422
15.8 FAQ 422
15.8.1 Windows 上远程打开 MindStudio 时，偶现复制的内容无法粘贴到编辑器窗口中 422
15.8.2 MobaXterm 启动 MindStudio 编辑字符时，偶现方向键变成数字 423
15.8.3 配置不受信任的网址访问浏览器 423
15.8.4 启动 MindStudio 报错：Failed to initialize graphics environment 424
15.8.5 TensorFlow 导出 Pb 模型 425
15.8.6 操作 MindStudio 时弹出 Unlock Keyring 窗口：Enter password to unlock 426
15.8.7 如何获取帮助和提出建议 427
15.9 修订记录 427

1 认识 MindStudio

1.1 关于MindStudio
1.2 What's New
1.3 关键概念
1.4 昇腾产品形态说明

1.1 关于 MindStudio

MindStudio是一套基于IntelliJ框架的开发工具链平台，提供了应用开发、调试、模型转换功能，同时还提供了网络移植、优化和分析功能，为用户开发应用程序带来了极大的便利。

说明

《MindStudio用户指南》需要根据产品选择其他相关手册适配使用，涉及安装以及开发相关辅助手册。

- Atlas 200 DK请参考《**Atlas 200 DK开发者套件**》。
- 其他产品请参考《**昇腾CANN**》。

功能简介

- 针对算子开发，MindStudio提供了全套的算子开发、调优能力。通过MindStudio提供的工具链也可以进行第三方算子开发，降低了算子开发的门槛，并提高算子开发及调试调优的效率。
- 针对网络模型的开发，MindStudio集成了离线模型转换工具、模型精度比对工具、模型运行性能分析工具、日志分析工具，提升了网络模型移植、分析和优化的效率。
- 针对应用开发，MindStudio集成了各种工具如分析器（Profiler）和编译器（Compiler）等，为开发者提供了图形化的集成开发环境，通过MindStudio能够进行工程管理、编译、调试、性能分析等全流程开发，能够很大程度提高开发效率。

功能框架

MindStudio功能框架如图1-1所示，目前含有的工具链包括：模型转换工具、模型训练工具、自定义算子开发工具、应用开发工具、工程管理工具、编译工具、流程编排工具、精度比对工具、日志管理工具、性能分析工具、设备管理工具等多种工具。

图 1-1 工具链功能架构

工具功能

MindStudio工具中的主要几个功能特性如下：

- 工程管理：为开发人员提供创建工程、打开工程、关闭工程、删除工程、新增工程文件目录和属性设置等功能。请参见13.2 工程管理。
- SSH管理：为开发人员提供新增SSH连接、删除SSH连接、修改SSH连接、复制SSH连接、加密SSH密码和修改SSH密码保存方式等功能。请参见13.1 SSH连接管理。
- 应用开发：针对业务流程开发人员，MindStudio工具提供基于AscendCL（Ascend Computing Language）和集成MindX SDK的应用开发编程方式，编程后的编译、运行、结果显示等一站式服务让流程开发更加智能化，可以让开发者快速上手。请参见7 应用开发。
- 自定义算子开发：提供了基于TBE的算子编程开发的集成开发环境，让不同平台下的算子移植更加便捷，适配昇腾AI处理器的速度更快。请参见8 自定义算子开发。
- 离线模型转换：训练好的第三方网络模型可以直接通过离线模型工具导入并转换成离线模型，并可一键式自动生成模型接口，方便开发者基于模型接口进行编程，同时也提供了离线模型的可视化功能。请参见4 模型转换。
- 日志管理：MindStudio为昇腾AI处理器提供了覆盖全系统的日志收集与日志分析解决方案，提升运行时算法问题的定位效率。提供了统一形式的跨平台日志可视化分析能力及运行时诊断能力，提升日志分析系统的易用性。请参见13.6 日志管理。

- 性能分析：MindStudio以图形界面呈现方式，实现针对主机和设备上多节点、多模块异构体系的高效、易用、可灵活扩展的系统化性能分析，以及针对昇腾AI处理器的性能和功耗的同步分析，满足算法优化对系统性能分析的需求。请参见**10 Profiling性能分析**。
- 设备管理：MindStudio提供设备管理工具，实现对连接到主机上的设备的管理功能。请参见**13.4 设备管理**。
- 精度比对：可以用来比对自有模型算子的运算结果与Caffe、TensorFlow、ONNX标准算子的运算结果，以便用来确认神经网络运算误差发生的原因。请参见**9 精度比对**。
- 开发工具包的安装与管理：为开发者提供基于昇腾AI处理器的相关算法开发套件包Ascend-cann-toolkit,旨在帮助开发者进行快速、高效的人工智能算法开发。开发者可以将开发套件包安装到MindStudio上，使用MindStudio进行快速开发，也可以使用独立的开发套件包进行开发。Ascend-cann-toolkit包含了基于昇腾AI处理器开发依赖的头文件和库文件、编译工具链、调优工具等。请参见**13.5 ADK管理**。

1.2 What's New

部署形态

- Ascend-cann-toolkit整包安装体验优化
- MindStudio新增支持Ubuntu-aarch64和Euler-aarch64操作系统
- MindStudio支持容器化部署方式

图 1-2 全流程开发工具链

算子开发

- 算子自动测试框架支持生成执行报告
- 算子工程ST测试支持将算子计算结果与标杆数据进行比对
- 支持MindSpore算子工程的创建与开发

图 1-3 算子开发

模型开发

- 集成MindSpore训练脚本执行功能
- 训练脚本支持一键转换
- 集成MindConverter训练脚本转换
- 支持对接MindInsight训练可视化
- 模型转换AIPP增强——动态AIPP支持多输入节点

版权所有 © 华为技术有限公司

图 1-4 MindConverter

应用开发

- 应用工程远端推送工程目录优化
- 支持本地安装MindX SDK软件包并集成到开发环境
- 支持可视化pipeline业务流编排工作

图 1-5 Pipeline 流程编排

开发辅助工具

- 新增系统级Profiling工具调优功能
- 支持TensorFlow子模型导出功能
- 支持模型局部结构和ONNX模型精度比对
- 支持SSH协议连接远端环境

图 1-6 SSH 连接

版权所有 © 华为技术有限公司

1.3 关键概念

Host/Device

在异构计算架构中，昇腾AI处理器与CPU通过PCIe总线连接在一起来协同工作：

- Host：CPU所在位置称为主机端（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服务器或者WindowsPC，利用昇腾AI处理器提供的NN（Neural-Network）计算能力完成业务。如图1-1中的开发平台层。
- 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。如图1-1中的芯片层。

对于Ascend RC产品形态，昇腾AI处理器所在硬件设备与与之相连接的ARM服务器合设，统称为Host。

ATC

训练好的第三方模型（Caffe、Tensorflow等）可通过ATC（Ascend Tensor Compiler）转换成昇腾AI处理器支持的离线模型，应用程序运行时，可通过GE进行离线模型的加载，进行推理。

离线模型功能架构如图1-7所示。

图 1-7 离线模型架构

GE：图引擎（Graph Engine），提供了Graph/Operator IR作为安全易用的构图接口集合，用户可以调用这些接口构建网络模型，设置模型所包含的图、图内的算子、以及模型和算子的属性。

DVPP

数字视觉预处理模块（Digital Vision Pre-Processing，DVPP），作为整个软件流执行过程中的的编解码和图像转换模块，为神经网络发挥着预处理辅助功能。当来自系统内存和网络的视频或图像数据进入昇腾AI处理器的计算资源中运算之前，由于系统对输入数据有固定的格式要求，如果数据未满足架构规定的输入格式、分辨率等要求，就需要调用数字视觉处理模块进行格式的转换，才可以进行后续的神经网络计算。

AIPP

AI预处理模块（AI Pre Process）是昇腾AI处理器提供的硬件图像预处理模块，包括色域转换，图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等功能。DVPP模块输出的图片多为对齐后的YUV420SP类型，不支持输出RGB图片。因此，业务流需要使用AIPP模块转换对齐后YUV420SP类型图片的格式，并抠出模型需要的输入图片。

YUV420SP

有损图像颜色编码格式，常用为YUV420SP_UV、YUV420SP_VU两种格式。

1.4 昇腾产品形态说明

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

- 昇腾 AI 处理器的工作模式如下：
  - 昇腾310 AI处理器有EP和RC两种模式。
  - 昇腾910 AI处理器只有EP模式。
- 支持RC模式的产品有：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模式的产品

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

  EP模式通常由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。

两种模式的产品及架构如**图1-8**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 1-8 RC 和 EP 模式

版权所有 © 华为技术有限公司

2 开发环境准备

2.1 安装介绍
2.2 安装MindStudio(物理机、msInstaller方式）
2.3 安装MindStudio(物理机、手动方式)
2.4 安装MindStudio(容器)

2.1 安装介绍

软件包介绍

- MindStudio所在的环境为开发环境，开发人员可以进行普通的工程管理、代码编写、编译、模型转换等功能；实际安装昇腾AI处理器的环境为运行环境，实际运行用户开发的应用。如果需要在真实的昇腾AI处理器上运行开发的工程，则需要将MindStudio连接到运行环境，并通过运行环境和带有昇腾AI处理器的设备上的工具后台服务模块进行配合，完成所有开发工程的运行、日志和性能分析等功能。
- Ascend-cann-toolkit：开发套件包。为开发者提供基于昇腾AI处理器的相关算法开发工具包，旨在帮助开发者进行快速、高效的模型、算子和应用的开发。开发套件包只能安装在Linux服务器上，开发者可以在安装开发套件包后，使用MindStudio开发工具进行快速开发。

使用场景

- (场景一)在昇腾AI设备或者非昇腾AI设备上安装Ascend-cann-toolkit开发套件包，可作为开发环境，仅能用于代码开发、编译等开发活动（例如ATC模型转换、算子和推理应用程序的纯代码开发）。如果想运行应用程序或进行训练，需要连接部署好环境的昇腾AI设备。
- (场景二)在昇腾AI设备上安装安装Ascend-cann-toolkit开发套件包、npu-firmware安装包、npu-driver安装包和AI框架，可以同时作为开发环境和运行环境，运行应用程序或进行训练。部署昇腾AI设备的详细操作请参见《**CANN 软件安装指南 (开发&运行场景，通过命令行方式)**》。

在 Linux 上安装

MindStudio可以安装在Linux服务器上，可以使用Linux服务器上原生桌面自带的终端gnome-terminal进行安装，也可以在Windows服务器上通过SSH登录到Linux服务器

进行安装，因为MindStudio是一款GUI程序，所以在Windows服务器上通过SSH登录到Linux服务器进行安装时，需要使用集成了X server的SSH终端（比如MobaXterm，该工具版本需要为v20.2及以上）。

- 通过msInstaller可自动安装操作系统依赖，以及MindStudio和开发套件包Ascend-cann-toolkit，详情请参考《**CANN 软件安装指南 (开发场景，通过msInstaller工具)**》。
- 通过手动方式部署开发环境请参考**2.3 安装MindStudio(物理机、手动方式)**。
- 通过容器方式部署开发环境需要通过网络拉取镜像文件，构建MindStudio容器。详情请参考**2.4 安装MindStudio(容器)**。

图 2-1 Linux 上安装方式架构图

2.2 安装 MindStudio(物理机、msInstaller 方式）

通过msInstaller可自动安装操作系统依赖，以及MindStudio和开发套件包Ascend-cann-toolkit，详情请参考《**CANN 软件安装指南 (开发场景，通过msInstaller工具)**》。

说明

- 如需通过手动方式安装MindStudio，请参考**2.3 安装MindStudio(物理机、手动方式)**。
- Atlas 200 DK的环境部署请参考《**Atlas 200 DK开发者套件**》的“环境部署”章节。

2.3 安装 MindStudio(物理机、手动方式)

2.3.1 安装流程

图 2-2 MindStudio 安装流程

2.3.2 获取软件包

表 2-1 软件包说明

软件包	说明	获取链接
MindStudio_{*version*}_linux.tar.gz	MindStudio软件包，含有GUI的集成开发环境。	获取软件包
Ascend-cann-toolkit_*{version}*_linux-*{arch}*.run	Ascend-cann-toolkit开发套件包，包含开发辅助工具和相关开发接口的开发套件包。 ● 若环境上已安装，则无需再次获取。 ● 若环境上未安装，可以获取该软件包后与MindStudio同步安装。	获取软件包

检查软件包的完整性

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。

2.3.3 安装前准备

2.3.3.1 Ubuntu-x86_64 系统

环境要求

安装MindStudio的环境，所要求的硬件及操作系统要满足以下条件。

表 2-2 Ubuntu 系统配套版本信息

类别	版本限制	说明
硬件	• 内存：最小4GB，推荐8GB • 磁盘空间：最小6GB	• 若Linux宿主机内存为4G，在MindStudio中进行模型转换时，建议Model文件大小不超过350M，如果超过此规格，操作系统可能会因为超过安全内存阈值而工作不稳定。 • 若Linux宿主机配置升级，比如8G内存，则相应支持的操作对象规格按比例提升。 例如，内存由4G升级到8G，则Model文件建议大小不超过700M。
操作系统	版本：18.04 64位X86操作系统	• 请从**http://releases.ubuntu.com/releases/**下载对应版本软件进行安装。 • 模型转换时，如果加载了TE自定义插件，可能会出现界面卡住的现象；如果界面卡住，请使用命令**uname -r**或者**cat /proc/version**检查Linux内核版本是否低于4.18，如果低于4.18请参见如下链接升级Linux内核补丁：**https://bugzilla.kernel.org/attachment.cgi?id=277305**，升级Linux内核的方法请参见：**https://bbs.huaweicloud.com/forum/thread-22441-1-1.html**。
系统语言	en_US.UTF-8	• 当前仅支持系统语言为英文。 • 请以任意用户使用**locale**命令在任意路径下查询编码格式，若系统返回“LANG=en_US.UTF-8”，则表示正确；否则，请以root用户使用“**vim /etc/default/locale**”命令修改“LANG=en_US.UTF-8”，重启（使用**reboot**命令）使之生效。

准备 MindStudio 安装用户（可选）

- 如果已安装Ascend-cann-toolkit开发套件包，请使用Ascend-cann-toolkit开发套件包的安装用户安装MindStudio。
- 如果未安装Ascend-cann-toolkit开发套件包，请参考如下示例准备安装用户。

您可以使用任意用户（含root或非root用户）安装MindStudio。

- 若使用root用户安装，则不需要操作该章节，不需要对root用户做任何设置。
- 若使用已存在的非root用户安装，须保证该用户对$HOME目录具有读写以及可执行权限。
- 若使用新的非root用户安装，请参考如下步骤进行创建，如下操作请在root用户下执行。本手册以该种场景为例执行MindStudio的安装。

  a. 执行以下命令创建MindStudio安装用户并设置该用户的$HOME目录。
  ```
  useradd -g HwHiAiUser -d /home/username -m username -s /bin/bash
  ```

  说明

  用户所属的属组必须和Driver运行用户所属属组相同；如果不同，请用户自行添加到Driver运行用户属组。例如以HwHiAiUser群组为例，可执行**useradd -g *HwHiAiUser* -d /home/*username* -m *username* -s /bin/bash**创建MindStudio安装用户并加入到群组中。

  b. 执行以下命令设置密码。
  ```
  passwd username
  ```
  *username*为安装MindStudio的用户名，该用户的umask值不能大于0027：

  - 若要查看umask的值，则执行命令：**umask**
  - 若要修改umask的值，则执行命令：**umask *新的取值***

    如果用户通过上述方式修改了umask取值，则修改后的取值只在当前窗口有效，用户也可以通过修改~/.bashrc文件方式设置永久umask取值：

    1) 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 *新的取值***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。

配置 MindStudio 安装用户权限（可选）

当用户使用非root用户安装时，需要操作该章节，否则请忽略。

MindStudio安装前需要下载相关依赖软件，可能用到以下提权命令，需要以root用户执行如下操作为普通用户提权。

步骤1 打开“/etc/sudoers”文件：
```
chmod u+w /etc/sudoers
vi /etc/sudoers
```

步骤2 在该文件中增加如下内容：
```
username ALL=(ALL:ALL)   NOPASSWD:SETENV:/usr/bin/apt-get, /usr/bin/pip, /bin/tar, /bin/mkdir, /bin/rm, /bin/sh, /bin/cp, /bin/bash, /usr/bin/make install, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /usr/bin/python3.7.5, /bin/ln, /bin/ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5, /bin/ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5, /usr/bin/unzip, /usr/sbin/update-alternatives, /usr/bin/add-apt-repository, /usr/local/python3.7.5/bin/pip3, /usr/bin/find /etc -name openssl.cnf
```

其中*username*代表安装用户，请根据实际替换。

说明

- 如果“/etc/sudoers”文件中没有“#includedir /etc/sudoers.d”，请在该文件最后一行手动添加该信息。
- 用户在完成MindStudio安装后，保留上述配置有提权风险，请在安装完成后立即取消sudo权限。取消方法为删除对应普通用户的所有白名单权限命令。
- 用户进行卸载操作时，如果提示权限相关问题，也需要配置以上sudo权限。卸载完成后请取消对应权限。

步骤3 添加完成后，执行**:wq!**保存文件。

步骤4 执行以下命令取消“/etc/sudoers”文件的写权限：

```
chmod u-w /etc/sudoers
```

----结束

检查源

MindStudio安装过程需要下载相关依赖，请确保安装MindStudio的服务器能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
apt-get update
```

说明

如果命令执行报错，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。

安装依赖

详见**15.2.1 安装依赖(Ubuntu-x86_64)**。

2.3.3.2 Ubuntu-aarch64 系统

环境要求

安装MindStudio的环境，所要求的硬件及操作系统要满足以下条件。

表 2-3 Ubuntu 系统配套版本信息

类别	版本限制	说明
硬件	• 内存：最小4GB，推荐8GB • 磁盘空间：最小6GB	• 若Linux宿主机内存为4G，在MindStudio中进行模型转换时，建议Model文件大小不超过350M，如果超过此规格，操作系统可能会因为超过安全内存阈值而工作不稳定。 • 若Linux宿主机配置升级，比如8G内存，则相应支持的操作对象规格按比例提升。 例如，内存由4G升级到8G，则Model文件建议大小不超过700M。

类别	版本限制	说明
操作系统	版本：18.04 64位ARM操作系统	● 模型转换时，如果加载了TE自定义插件，可能会出现界面卡住的现象；如果界面卡住，请使用命令**uname -r**或者**cat /proc/version**检查Linux内核版本是否低于4.18，如果低于4.18请参见如下链接升级Linux内核补丁：**https://bugzilla.kernel.org/attachment.cgi?id=277305** ● 升级Linux内核的方法请参见：**https://bbs.huaweicloud.com/forum/thread-22441-1-1.html**。
系统语言	en_US.UTF-8	● 当前仅支持系统语言为英文。 ● 请以任意用户使用**locale**命令在任意路径下查询编码格式，若系统返回“LANG=en_US.UTF-8”，则表示正确；否则，请以root用户使用“**vim /etc/default/locale**”命令修改“LANG=en_US.UTF-8”，重启（使用**reboot**命令）使之生效。

准备 MindStudio 安装用户（可选）

- 如果已安装Ascend-cann-toolkit开发套件包，请使用Ascend-cann-toolkit开发套件包的安装用户安装MindStudio。
- 如果未安装Ascend-cann-toolkit开发套件包，请参考如下示例准备安装用户。

您可以使用任意用户（含root或非root用户）安装MindStudio。

- 若使用root用户安装，则不需要操作该章节，不需要对root用户做任何设置。
- 若使用已存在的非root用户安装，须保证该用户对$HOME目录具有读写以及可执行权限。
- 若使用新的非root用户安装，请参考如下步骤进行创建，如下操作请在root用户下执行。本手册以该种场景为例执行MindStudio的安装。
  a. 执行以下命令创建MindStudio安装用户并设置该用户的$HOME目录。
  ```
  useradd -g HwHiAiUser -d /home/username -m username -s /bin/bash
  ```
  说明

  用户所属的属组必须和Driver运行用户所属属组相同；如果不同，请用户自行添加到Driver运行用户属组。例如以HwHiAiUser群组为例，可执行**useradd -g *HwHiAiUser* -d /home/*username* -m *username* -s /bin/bash**创建MindStudio安装用户并加入到群组中。

  b. 执行以下命令设置密码。
  ```
  passwd username
  ```
  *username*为安装MindStudio的用户名，该用户的umask值不能大于0027：
  - 若要查看umask的值，则执行命令：**umask**
  - 若要修改umask的值，则执行命令：**umask *新的取值***

    如果用户通过上述方式修改了umask取值，则修改后的取值只在当前窗口有效，用户也可以通过修改~/.bashrc文件方式设置永久umask取值：

1) 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** ***新的取值***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。

配置 MindStudio 安装用户权限（可选）

当用户使用非root用户安装时，需要操作该章节，否则请忽略。

MindStudio安装前需要下载相关依赖软件，可能用到以下提权命令，需要以root用户执行如下操作为普通用户提权。

步骤1 打开“/etc/sudoers”文件：
```
chmod u+w /etc/sudoers
vi /etc/sudoers
```

步骤2 在该文件中增加如下内容：
```
username ALL=(ALL:ALL)  NOPASSWD:SETENV:/usr/bin/apt-get, /usr/bin/pip, /bin/tar, /bin/mkdir, /bin/rm, /bin/sh, /bin/cp, /bin/bash, /usr/bin/make install, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /usr/bin/python3.7.5, /bin/ln, /bin/ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5, /bin/ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5, /usr/bin/unzip, /usr/sbin/update-alternatives, /usr/bin/add-apt-repository, /usr/local/python3.7.5/bin/pip3, /usr/bin/find /etc -name openssl.cnf
```

其中*username*代表安装用户，请根据实际替换。

说明

- 如果“/etc/sudoers”文件中没有“#includedir /etc/sudoers.d”，请在该文件最后一行手动添加该信息。
- 用户在完成MindStudio安装后，保留上述配置有提权风险，请在安装完成后立即取消sudo权限。取消方法为删除对应普通用户的所有白名单权限命令。
- 用户进行卸载操作时，如果提示权限相关问题，也需要配置以上sudo权限。卸载完成后请取消对应权限。

步骤3 添加完成后，执行**:wq!**保存文件。

步骤4 执行以下命令取消“/etc/sudoers”文件的写权限：
```
chmod u-w /etc/sudoers
```

----结束

检查源

MindStudio安装过程需要下载相关依赖，请确保安装MindStudio的服务器能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
apt-get update
```

说明

如果命令执行报错，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。

安装依赖

详见15.2.2 安装依赖(Ubuntu-aarch64)。

2.3.3.3 EulerOS-aarch64 系统

环境要求

安装MindStudio的环境，所要求的硬件及操作系统要满足以下条件。

表 2-4 系统配套版本信息

类别	版本限制	说明
硬件	● 内存：最小4GB，推荐8GB ● 磁盘空间：最小6GB	● 若Linux宿主机内存为4G，在MindStudio中进行模型转换时，建议Model文件大小不超过350M，如果超过此规格，操作系统可能会因为超过安全内存阈值而工作不稳定。 ● 若Linux宿主机配置升级，比如8G内存，则相应支持的操作对象规格按比例提升。 例如，内存由4G升级到8G，则Model文件建议大小不超过700M。
操作系统	版本：2.8 64位ARM操作系统	--
系统语言	en_US.UTF-8	● 当前仅支持系统语言为英文。 ● 请以任意用户使用**locale**命令在任意路径下查询编码格式，若系统返回“LANG=en_US.UTF-8”，则表示正确；否则，请以root用户使用“**vim /etc/default/locale**”命令修改“LANG=en_US.UTF-8”，重启（使用**reboot**命令）使之生效。

准备 MindStudio 安装用户（可选）

- 如果已安装Ascend-cann-toolkit开发套件包，请使用Ascend-cann-toolkit开发套件包的安装用户安装MindStudio。
- 如果未安装Ascend-cann-toolkit开发套件包，请参考如下示例准备安装用户。

您可以使用任意用户（含root或非root用户）安装MindStudio。

- 若使用root用户安装，则不需要操作该章节，不需要对root用户做任何设置。
- 若使用已存在的非root用户安装，须保证该用户对$HOME目录具有读写以及可执行权限。
- 若使用新的非root用户安装，请参考如下步骤进行创建，如下操作请在root用户下执行。本手册以该种场景为例执行MindStudio的安装。

  a. 执行以下命令创建MindStudio安装用户并设置该用户的$HOME目录。
  ```
  useradd -g HwHiAiUser -d /home/username -m username -s /bin/bash
  ```

说明

用户所属的属组必须和Driver运行用户所属属组相同；如果不同，请用户自行添加到Driver运行用户属组。例如以HwHiAiUser群组为例，可执行**useradd -g *HwHiAiUser* -d /home/*username* -m *username* -s /bin/bash**创建MindStudio安装用户并加入到群组中。

b. 执行以下命令设置密码。

```
passwd username
```

*username*为安装MindStudio的用户名，该用户的umask值不能大于0027：

- 若要查看umask的值，则执行命令：**umask**
- 若要修改umask的值，则执行命令：**umask *新的取值***

  如果用户通过上述方式修改了umask取值，则修改后的取值只在当前窗口有效，用户也可以通过修改~/.bashrc文件方式设置永久umask取值：

  1) 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 *新的取值***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。

配置 MindStudio 安装用户权限（可选）

当用户使用非root用户安装时，需要操作该章节，否则请忽略。

MindStudio安装前需要下载相关依赖软件，可能用到以下提权命令，需要以root用户执行如下操作为普通用户提权。

1. 打开“/etc/sudoers”文件：

```
chmod u+w /etc/sudoers
vi /etc/sudoers
```

2. 在该文件"root ALL=(ALL:ALL) ALL"下面增加如下内容：

```
username ALL=(ALL:ALL)  NOPASSWD:SETENV:/usr/bin/yum, /usr/bin/pip, /bin/tar, /bin/mkdir, /bin/rm, /bin/sh, /bin/cp, /bin/bash, /usr/bin/make install, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /usr/bin/python3.7.5, /bin/ln, /bin/ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5, /bin/ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5, /usr/bin/unzip, /usr/sbin/update-alternatives, /usr/bin/add-apt-repository, /usr/local/python3.7.5/bin/pip3, /usr/bin/find /etc -name openssl.cnf
```

其中*username*代表安装用户，请根据实际替换。

说明

- 如果“/etc/sudoers”文件中没有“#includedir /etc/sudoers.d”，请在该文件最后一行手动添加该信息。
- 用户在完成MindStudio安装后，保留上述配置有提权风险，请在安装完成后立即取消sudo权限。取消方法为删除对应普通用户的所有白名单权限命令。
- 用户进行卸载操作时，如果提示权限相关问题，也需要配置以上sudo权限。卸载完成后请取消对应权限。

3. 添加完成后，执行**:wq!**保存文件。
4. 执行以下命令取消“/etc/sudoers”文件的写权限：

```
chmod u-w /etc/sudoers
```

检查源

MindStudio安装过程需要下载相关依赖，请确保安装MindStudio的服务器能够连接网络。

请在root用户下执行如下命令检查源是否可用。

```
yum repolist
```

说明

如果命令执行报错，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。

安装依赖

详见**15.2.3 安装依赖(EulerOS-aarch64)**。

2.3.4 安装 MindStudio

须知

- 当前不支持在同一服务器的同一目录安装多个MindStudio，如果系统上同一目录下已存在MindStudio，请参见**14.5 卸载MindStudio**章节，将原MindStudio卸载后再执行安装。
- 请使用MindStudio的安装用户登录，不支持使用其他用户登录后再切换到MindStudio的安装用户。

前提条件

已完成**2.3.2 获取软件包**和**2.3.3 安装前准备**。

操作步骤

步骤1 使用MindStudio的安装用户上传软件包。

- 已安装Ascend-cann-toolkit开发套件包：将MindStudio软件包上传至MindStudio安装服务器。
- 未安装Ascend-cann-toolkit开发套件包：将MindStudio软件包和Ascend-cann-toolkit开发套件包上传至MindStudio安装服务器。

步骤2 解压MindStudio软件包。

使用MindStudio的安装用户在软件包所在路径执行如下命令，解压MindStudio_{*version*}_linux.tar.gz软件包。

```
tar -zxvf MindStudio_{version}_linux.tar.gz
```

解压后包的内容以及说明请参见**表15-1**。

步骤3 使用MindStudio的安装用户进入软件包解压后的MindStudio/bin目录，执行如下命令：

```
cd MindStudio/bin
./MindStudio.sh
```

说明

如果启动MindStudio报错：Failed to initialize graphics environment，请参见**15.8.4 启动MindStudio报错：Failed to initialize graphics environment**。

首次启动时，MindStudio会引导用户安装所需第三方依赖文件，建议用户不要使用后台运行的方式启动MindStudio。安装脚本会自动检查该路径是否有效：

1. 如果路径有效，则检查是否安装相关依赖，如果所有依赖已经安装，则进入**步骤4**；否则参见提示安装缺失的依赖软件，安装完成后重新执行安装脚本。
2. 如果提示如下信息：
   ```
   There is no python3.7.5 in path /usr/local/python3.7.5.Please enter your choice:
   1.choose local python3.7.5
   2.install python3.7.5
   ```
   a. 输入**1**后提示“Enter local python path:”，输入本地已经安装python3.7.5的路径，然后脚本会检查该路径是否有效：
      i. 若该路径检查通过，则检查是否安装相关依赖，若所有依赖已经安装，则进入**步骤4**；否则按照提示安装相关依赖，安装完成后重新执行安装脚本。
      ii. 若路径检查不通过，则返回**提示信息**处，重新检查输入路径是否正确或输入**2**，进入**b**。
   b. 输入**2**，根据如下界面提示信息，输入新的安装路径安装python3.7.5（不输入默认新的安装路径为/usr/local/python3.7.5）：
      ```
      Enter python install path.Default value is /usr/local/python3.7.5:
      /home/hisisoc/software/python3.7.5
      ```
      按照提示信息安装python3.7.5以及相关依赖，安装完成后重新执行安装脚本。

步骤4 进入导入设置界面，如**图2-3**所示界面。

图 2-3 环境设置

- Config or installation folder：表示从自定义配置目录导入MindStudio设置。
  自定义配置目录所在路径：“$HOME/.config/Huawei/MindStudioMS-x.x”，MindStudio设置包括工程界面的个性化设置（例如背景色等信息）等。
- Do not import settings：不导入设置，若选择该选项，则创建新的配置文件，默认为该选项。

根据需要选择相关选项后，单击“OK”，进入下一步。

说明

- 若已经安装过低版本MindStudio，再次安装高版本时，则图2-3界面中也会出现“Previous version”参数，表示是否从低版本导入MindStudio设置，您可以根据需要进行选择。
- 若已经安装过相同版本MindStudio，且卸载前一版本时未删除$HOME/.cache/Huawei/MindStudioMS-*{version}*目录和$HOME/.config/Huawei/MindStudioMS-*{version}*，则安装下个版本时会从步骤步骤3直接进入步骤9。

步骤5 系统如果弹出如图2-4所示授权信息界面。

说明

在使用MindStudio过程中也会弹出此界面，可根据情况选择是否接受。在菜单栏选择File > Settings... > Tools > Server Certificates，勾选“Accept non-trusted certificates automatically”后即可自动接受，在使用MindStudio过程中不再弹出授权提示信息。

图 2-4 授权提示信息

- 单击“Accept”表示同意MindStudio连接升级服务器。
- 单击“Reject”表示拒绝MindStudio连接升级服务器，下次重新启动MindStudio时还会弹出该界面。

步骤6 弹出图2-5所示界面。

说明

选择Ascend-cann-toolkit开发套件包安装路径时，请不要选择安装开发套件包时系统自动创建的软链接路径。

图 2-5 选择 Ascend-cann-toolkit 开发套件包路径

- 已经安装Ascend-cann-toolkit开发套件包：单击“Ascend Toolkit Path”选项后面的，在弹出界面中选择Ascend-cann-toolkit开发套件包的安装路径（需要选择到版本号一级），单击“OK”后，“Ascend Toolkit Version”会显示选择的Ascend-cann-toolkit开发套件包版本信息，不能修改，如图2-6所示。单击“Finish”，进入步骤9。

图 2-6 选择 Ascend-cann-toolkit 开发套件包安装路径

- 未安装Ascend-cann-toolkit开发套件包：单击“Ascend Toolkit Path”选项后面的，在弹出界面中选择**2.3.3 安装前准备**中上传的Ascend-cann-toolkit开发套件包，并选择其所在路径，如**图2-7**所示。单击Next，进入**步骤7**。

图 2-7 选择 Ascend-cann-toolkit 开发套件包所在路径

步骤7 进入安装信息确认界面，检查当前安装信息是否正确。

版权所有 © 华为技术有限公司

图 2-8 安装信息确认

说明

若用户之前在**Install Location**路径安装过相同版本的Ascend-cann-toolkit开发套件包，则弹出信息提示已经安装。

步骤8 确认信息正确后，单击“Next ”解压并安装相关软件包，如图2-9所示。

图 2-9 安装

单击“Show Details”可以查看Ascend-cann-toolkit开发套件包安装详细信息，单击“Hide Details”隐藏相关信息。

若出现如下信息表明安装成功，此时“Finish”按钮变成可单击状态，其余按钮置灰。

```
Installation succeeded. Please click "Finish".
```

单击“Finish”，弹出如图2-10所示重启MindStudio使安装生效界面。

图 2-10 重启提示信息

单击“OK”，重新启动MindStudio。

步骤9 如果没有报错信息且能正常进入欢迎界面，则表示MindStudio安装成功。

安装成功后，MindStudio安装服务器$HOME路径会出现如下目录：

- $HOME/.mindstudio：包括Toolkit安装路径（mindstudio.properties文件）、MindStudio属性文件以及隐私声明界面用户所做的操作信息。
- $HOME/.cache/Huawei/MindStudioMS-*{version}*：存储MindStudio安装以及启动后的运行日志（log/idea.log）等信息。
- $HOME/.config/Huawei/MindStudioMS-*{version}*：存储用户配置数据目录，包括工程界面的个性化设置（例如背景色等信息）等信息。

欢迎界面参数解释如下：

- Create New Project：创建新工程，创建后工程保存在“$HOME/AscendProjects”目录。
- Open or Import：打开或导入已有工程。
- Welcome System Profiling：进入System Profiling界面。

步骤10 启动MindStudio，在菜单栏选择File > Settings... > Plugins，搜索并安装如下插件工具。

- Grep Console
- Pylint
- Python Community Edition

----结束

2.3.5 配置编译环境

安装完MindStudio并进入相关工程页面后，如果用户使用算子工程或应用工程开发了相关程序，需要进行工程的编译和运行，用于生成相关二进制文件，进行工程编译前，需要配置编译环境。

安装编译工具

根据产品形态以及运行环境对应操作系统及架构，安装不同的编译工具。

- Ascend EP产品形态

表 2-5 编译工具使用命令

MindStudio安装服务器环境	运行环境操作系统以及架构	编译工具
ubuntu18.04+x86	CentOS7.6+x86	请使用MindStudio安装用户，在MindStudio安装服务器执行**g++ --version**命令检查是否安装，若已经安装则可以忽略，否则请执行如下安装命令： `sudo apt-get install -y g++` 如果用户MindStudio安装服务器存在多个交叉编译器，则请用户自行将所用的交叉编译器软链接到上述要求的交叉编译器版本上。命令如下： `sudo ln -s` *其他版本的交叉编译器 环境准备所要求的版本*
	CentOS7.6+Arm64	请使用MindStudio安装用户，在MindStudio安装服务器执行**aarch64-linux-gnu-g++ --version**命令检查是否安装，若已经安装则可以忽略，否则请执行如下安装命令： `sudo apt-get install -y g++-aarch64-linux-gnu` 如果用户MindStudio安装服务器存在多个交叉编译器，则请用户自行将所用的交叉编译器软链接到上述要求的交叉编译器版本上。命令如下： `sudo ln -s` *其他版本的交叉编译器 环境准备所要求的版本*

	EulerOS2.5+x86	如下命令需要逐行执行。
		1. 增加**g++**安装源。
		`sudo apt-get install -y software-properties-common`
		`sudo -E add-apt-repository ppa:ubuntu-toolchain-r/test`
		2. 更新源。
		`sudo -E apt-get update`
		3. 安装**gcc-7**和**g++-7**。
		`sudo -E apt-get install -y gcc-7 --fix-missing`
		`sudo -E apt-get install -y g++-7 --fix-missing`
		4. 更新软链接。
		`sudo update-alternatives --install /usr/bin/gcc-7.4 gcc-7.4 /usr/bin/gcc-7 100`
		`sudo update-alternatives --config gcc-7.4`
		`sudo update-alternatives --install /usr/bin/g++-7.4 g++-7.4 /usr/bin/g++-7 100`
		`sudo update-alternatives --config g++-7.4`
		`sudo update-alternatives --install /usr/bin/gcc gcc-7.4 /usr/bin/gcc-7 100`
		`sudo update-alternatives --config gcc`
		`sudo update-alternatives --install /usr/bin/g++ g++-7.4 /usr/bin/g++-7 100`
		`sudo update-alternatives --config g++`

- Ascend RC产品形态

表 2-6 编译工具使用命令

MindStudio安装服务器环境	运行环境操作系统以及架构	编译工具
ubuntu18.04+x86	ubuntu18.04+aarch64	请使用MindStudio安装用户，在MindStudio安装服务器执行**aarch64-linux-gnu-g++ --version**命令检查是否安装，若已经安装则可以忽略，否则请执行如下安装命令： sudo apt-get install g++-aarch64-linux-gnu 如果用户MindStudio安装服务器存在多个交叉编译器，则请用户自行将所用的交叉编译器软链接到上述要求的交叉编译器版本上。命令如下： sudo ln -s *其他版本的交叉编译器 要求的版本*

2.4 安装 MindStudio(容器)

2.4.1 安装流程

图 2-11 MindStudio 容器场景安装流程

2.4.2 安装前准备

- 宿主机需要已安装docker，可执行**docker --version**来检查。
- 安装用户需要加入docker组，本文以安装用户为**HwHiAiUser**为例，可执行**usermod -a -G docker *HwHiAiUser***将HwHiAiUser加入docker组。在某些环境，请执行**chmod 666 /var/run/docker.sock**以保证安装用户有权限操作docker。

- 宿主机拉取镜像需要联网，若未联网，请参见**15.6 配置系统网络代理**。
- 请按照**表2-7**所示，获取软件包，并拷贝到宿主机的某个目录下。

表 2-7 所需软件

软件包	说明	获取方法
MindStudio_{*version*}_linux.tar.gz	MindStudio软件包。	**获取链接**
Ascend-cann-toolkit_*{version}*_linux-*{arch}*.run	Ascend-cann-toolkit开发套件包。	**获取链接**
mindspore_ascend-{version}-cp37-cp37m-linux_{arch}.whl	MindSpore安装包。	仅训练场景需要。 **获取链接**
mindinsight-{version}-cp37-cp37m-linux_{arch}.whl	MindInsight安装包。	仅训练场景需要。 **获取链接**
matplotlib-3.3.3.tar.gz	Matplotlib 是 Python 的绘图库。 它可与 NumPy 一起使用，提供了一种有效的 MatLab 开源替代方案。	**获取链接**
ascend_install.info	软件包安装日志文件。	仅训练场景和使用AICPU算子进行推理调测场景需要。 获取方法：从host拷贝/etc/ascend_install.info，以实际路径为准
version.info	driver包版本信息文件。	仅训练场景和使用AICPU算子进行推理调测场景需要。 获取方法：从host拷贝/usr/local/Ascend/driver/version.info，以实际路径为准
gmp-6.1.2.tar.xz	GMP(The GNU Multiple Precision Arithmetic Library)又叫GNU多精度算术库，是一个提供了很多操作高精度的大整数，浮点数的运算的算术库，几乎没有什么精度方面的限制，功能丰富。	仅训练场景需要。 **获取链接**

宿主机要求

表 2-8 宿主机要求

类别	要求	说明
硬件	● 内存：最小4GB，推荐8GB ● 磁盘空间：最小30G	--

宿主机与容器操作系统兼容性关系

宿主机操作系统与容器操作系统的兼容性关系如**表2-9**所示。

表 2-9 兼容性关系

架构	宿主机OS	容器OS版本
aarch64	--	Euler 2.8
		Ubuntu 18.04

2.4.3 构建 MindStudio 容器镜像

2.4.3.1 使用 Dockerfile 定制镜像

使用Dockerfile构建容器镜像，用户可根据实际需要在基础镜像Dockerfile文件基础上进行二次开发。

获取 Dockerfile 示例文件

步骤1 解压MindStudio软件包。

使用安装用户在软件包所在路径执行如下命令，解压MindStudio_{*version*}_linux.tar.gz软件包

```
tar -zxvf MindStudio_{version}_linux.tar.gz
```

步骤2 进入解压后的目录，选择您需要的Dockerfile示例文件。

```
cd ./MindStudio
```

目录结构	文件说明
./docker/infer/aarch64-linux/euler/Dockerfile	适用于推理、ARM架构，euler2.8的Dockerfile示例文件
./docker/infer/aarch64-linux/ubuntu/Dockerfile	适用于推理、ARM架构，ubuntu18.04的Dockerfile示例文件

目录结构	文件说明
./docker/train/aarch64-linux/euler/Dockerfile	适用于训练、ARM架构，euler2.8的Dockerfile示例文件
./docker/train/aarch64-linux/ubuntu/Dockerfile	适用于训练、ARM架构，ubuntu18.04的Dockerfile示例文件

----结束

使用 Dockerfile 定制镜像的操作步骤

步骤1 参考**获取Dockerfile示例文件**，根据实际情况定制修改Dockerfile，**表2-10**中列出的是必须修改点，其他的请根据需要修改。

表 2-10 Dockerfile 修改点

参数	参数说明	备注
CANN_TOOLKIT_NAME	Ascend-cann-toolkit的run包名称	修改等号后面参数值为**实际获取到的**包名称。
MINDSTUDIO_NAME	MindStudio软件包名称	修改等号后面参数值为**实际获取到的**包名称。
MINDSPORE_NAME	MindSpore安装包名称	只训练场景涉及。 修改等号后面参数值为**实际获取到的**包名称。
MINDINSIGHT_NAME	MindInsight安装包名称	只训练场景涉及。 修改等号后面参数值为**实际获取到的**包名称。

步骤2 按**表2-11**拷贝依赖的文件到Dockerfile 文件所在目录。

表 2-11 Dockerfile 依赖的文件

场景	依赖的文件
构建基于aarch64的MindStudio推理容器镜像	1. MindStudio_{*version*}_linux.tar.gz 2. Ascend-cann-toolkit_*{version}*_linux-*{arch}*.run 3. matplotlib-3.3.3.tar.gz 4. ascend_install.info（可选，使用AICPU算子进行推理调测时需要） 5. version.info（可选，使用AICPU算子进行推理调测时需要）

场景	依赖的文件
构建基于aarch64的MindStudio训练容器镜像	1. MindStudio_{*version*}_linux.tar.gz 2. Ascend-cann-toolkit_*{version}*_linux-*{arch}*.run 3. mindspore_ascend-{version}-cp37-cp37m-linux_{arch}.whl 4. mindinsight-{version}-cp37-cp37m-linux_{arch}.whl 5. matplotlib-3.3.3.tar.gz 6. ascend_install.info 7. version.info 8. gmp-6.1.2.tar.xz

步骤3 （可选）如果需要使用AICPU算子进行推理调测，请在对应的Dockerfile的“COPY matplotlib-3.3.3.tar.gz /root”行后面增加如下两行。

```
COPY ascend_install.info /etc/
COPY version.info /usr/local/Ascend/driver/
```

步骤4 在 Dockerfile 文件所在目录下，执行构建动作。

```
docker build -t image_name .
```

注：最后的.代表本次执行的上下文路径，请不要遗漏；*image_name请*替换为"镜像名称:镜像标签"

当出现“**Successfully built**”表示镜像构建成功。

步骤5 镜像构建完成后，执行以下命令查看镜像信息。

docker images

显示示例：

```
REPOSITORY      TAG         IMAGE ID        CREATED         SIZE
mindstudio      v1          d82746acd7f0    27 minutes ago  10.2G
```

说明

构建镜像过程中会生成一个名为“none”的中间镜像，为避免占用空间，建议删除。

----结束

2.4.4 启动 MindStudio 容器

2.4.4.1 在昇腾 AI 设备上启动 MindStudio 容器

须知

在昇腾AI设备上启动MindStudio容器，需要已在宿主机上安装固件&驱动。

步骤1 以安装用户登录宿主机，并导入环境变量，环境变量的值需要根据实际情况修改。

举例如下：

```
# 宿主机上的NPU设备，比如/dev/davinci0，/dev/davinci1，/dev/davinci2等
export device=/dev/davinci0
```

```
# 容器中的SSH端口（22）映射到主机的端口（20000），主机端口需要未被占用
export port=20000
# 需要设置的DISPLAY环境变量
export DISPLAY=显示图形界面的主机IP:0
# 构建的MindStudio镜像名称及标签
export image_name=mindstudio:v1
# 运行的容器名称
export container_name=mindstudio
```

步骤2 在宿主机上执行如下命令，基于已构建的MindStudio镜像运行一个容器。（下面命令中的变量已在**步骤1**中设置）

```
docker run -itd \
    --device=${device} \
    --device=/dev/davinci_manager \
    --device=/dev/devmm_svm \
    --device=/dev/hisi_hdc \
    --cap-add=SYS_PTRACE --cap-add=SYS_ADMIN \
    -v /sys/kernel/debug:/sys/kernel/debug \
    -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
    -v $HOME/.Xauthority:/root/.Xauthority \
    -e DISPLAY=$DISPLAY \
    -p ${port}:22 \
    --name ${container_name} \
    ${image_name}
```

表 2-12 参数解释

参数	参数说明
-itd	-i：以交互模式运行容器 -t：为容器重新分配一个伪输入终端 -d：后台运行容器，并返回容器ID
--device=XXX	将宿主机上的设备(NPU设备和管理设备)映射到容器中。请根据实际情况修改。
--cap-add=SYS_PTRACE --cap-add=SYS_ADMIN	在容器内启动System-Profiling性能分析工具需要的权限。
-v 宿主机目录:容器目录	将宿主机上的目录挂载到容器中。
-e DISPLAY=$DISPLAY	设置容器中的DISPLAY环境变量为宿主机上的DISPLAY环境变量。
-p *${port}*:22	将容器SSH端口22映射到宿主机的某个端口，宿主机上的端口不能被占用，可执行“netstat -anp |grep 端口号 ”，如果没有回显，表示端口未被占用
--name *${container_name}*	为容器指定一个名称
${image_name}	运行容器使用的镜像。格式为“镜像名称:镜像标签”，请根据实际情况修改

执行该命令后，如果显示容器ID，则表示mindstudio容器已经运行。

步骤3 进入容器。

方法一：通过docker exec进入容器

```
docker exec -it ${container_name} /bin/bash
```

方法二：通过SSH进入容器

```
# 初次始用时，请先通过docker exec进入容器后执行passwd，以设置root用户密码
ssh root@{宿主机IP} -p ${port}
```

步骤4 （可选）在容器中启动System Profiling。

```
cd /usr/local/Ascend/ascend-toolkit/latest/toolkit/tools/system-profiling/script/
./start.sh
```

步骤5 在容器中启动MindStudio。

```
cd /root/MindStudio/bin
./MindStudio.sh
```

首次启动时，MindStudio会交互式引导用户安装所需第三方依赖文件，建议用户不要使用后台运行的方式启动MindStudio。安装脚本会自动检查该路径是否有效：

1. 如果路径有效，则检查是否安装相关依赖，如果所有依赖已经安装，则进入**步骤6**；否则参见提示安装缺失的依赖软件，安装完成后重新执行安装脚本。
2. 如果提示如下信息：

   ```
   There is no python3.7.5 in path /usr/local/python3.7.5.Please enter your choice:
   1.choose local python3.7.5
   2.install python3.7.5
   ```

   a. 输入**1**后提示“Enter local python path:”，输入本地已经安装python3.7.5的路径，然后脚本会检查该路径是否有效：

      i. 若该路径检查通过，则检查是否安装相关依赖，若所有依赖已经安装，则进入**步骤6**；否则按照提示安装相关依赖，安装完成后重新执行安装脚本。

      ii. 若路径检查不通过，则返回**提示信息**处，重新检查输入路径是否正确或输入**2**，进入**b**。

   b. 输入**2**，根据如下界面提示信息，输入新的安装路径安装python3.7.5（不输入默认新的安装路径为/usr/local/python3.7.5）：

      ```
      Enter python install path.Default value is /usr/local/python3.7.5:
      /home/hisisoc/software/python3.7.5
      ```

      按照提示信息安装python3.7.5以及相关依赖，安装完成后重新执行安装脚本。

步骤6 进入导入设置界面，如**图2-12**所示界面。

图 2-12 环境设置

- Config or installation folder：表示从自定义配置目录导入MindStudio设置。

  自定义配置目录所在路径：“$HOME/.config/Huawei/MindStudioMS-x.x”，MindStudio设置包括工程界面的个性化设置（例如背景色等信息）等。
- Do not import settings：不导入设置，若选择该选项，则创建新的配置文件，默认为该选项。

根据需要选择相关选项后，单击“OK”，进入下一步。

步骤7 系统弹出如图2-13所示授权信息界面。

图 2-13 授权提示信息

- 单击“Accept”表示同意MindStudio连接升级服务器。
- 单击“Reject”表示拒绝MindStudio连接升级服务器，下次重新启动MindStudio时还会弹出该界面。

步骤8 弹出图2-14所示界面。

说明

选择Ascend-cann-toolkit开发套件包安装路径时，请不要选择安装开发套件包时系统自动创建的软链接路径。

图 2-14 选择 Ascend-cann-toolkit 开发套件包路径

- 已经安装Ascend-cann-toolkit开发套件包：单击“Ascend Toolkit Path”选项后面的，在弹出界面中选择Ascend-cann-toolkit开发套件包的安装路径（需要选择到版本号路径），单击“OK”后，“Ascend Toolkit Version”会显示选择的Ascend-cann-toolkit开发套件包版本信息，不能修改，如图2-15所示。单击“Finish”，进入步骤11。

图 2-15 选择 Ascend-cann-toolkit 开发套件包安装路径

版权所有 © 华为技术有限公司

- 未安装Ascend-cann-toolkit开发套件包：单击“Ascend Toolkit Path”选项后面的，在弹出界面中选择表2-7中上传的Ascend-cann-toolkit开发套件包，并选择其所在路径，如图2-16所示。单击Next，进入步骤9。

图 2-16 选择 Ascend-cann-toolkit 开发套件包所在路径

步骤9 进入安装信息确认界面，检查当前安装信息是否正确。

图 2-17 安装信息确认

说明

若用户之前在**Install Location**路径安装过相同版本的Ascend-cann-toolkit开发套件包，则弹出信息提示已经安装。

步骤10 确认信息正确后，单击“Next ”解压并安装相关软件包，如图2-18所示。

图 2-18 安装

单击“Show Details”可以查看Ascend-cann-toolkit开发套件包安装详细信息，单击“Hide Details”隐藏相关信息。

若出现如下信息表明安装成功，此时“Finish”按钮变成可单击状态，其余按钮置灰。

```
Installation succeeded. Please click "Finish".
```

单击“Finish”，弹出如图2-19所示重启MindStudio使安装生效界面。

图 2-19 重启提示信息

单击“OK”，重新启动MindStudio。

步骤11 如果没有报错信息且能正常进入欢迎界面，则表示MindStudio安装成功。

安装成功后，MindStudio安装服务器$HOME路径会出现如下目录：

- $HOME/.mindstudio：包括Toolkit安装路径（mindstudio.properties文件）、MindStudio属性文件以及隐私声明界面用户所做的操作信息。

- $HOME/.cache/Huawei/MindStudioMS-*{version}*：存储MindStudio安装以及启动后的运行日志（log/idea.log）等信息。
- $HOME/.config/Huawei/MindStudioMS-*{version}*：存储用户配置数据目录，包括工程界面的个性化设置（例如背景色等信息）等信息。

欢迎界面参数解释如下：

- Create New Project：创建新工程，创建后工程保存在“$HOME/AscendProjects”目录。
- Open or Import：打开或导入已有工程。
- Welcome System Profiling：进入System Profiling界面。

步骤12 启动MindStudio，在菜单栏选择File > Settings... > Plugins，搜索并安装如下插件工具。

- Grep Console
- Pylint
- Python Community Edition

----结束

2.4.4.2 在非昇腾 AI 设备上启动 MindStudio 容器

须知

在非昇腾AI设备上启动MindStudio容器与在昇腾AI设备上启动MindStudio容器相比，除步骤1导入环境变量和步骤2容器运行命令不同外，其它步骤均相同。

步骤1 在宿主机导入环境变量，环境变量的值需要根据实际情况修改。

举例如下：

```
# 容器中的SSH端口（22）映射到主机的端口（20000），主机端口需要未被占用
export port=20000
# X11需要设置的DISPLAY环境变量
export DISPLAY=显示图形界面的XServer主机IP:0
# 构建的MindStudio镜像名称及标签
export image_name=mindstudio:v1
# 运行的容器名称
export container_name=mindstudio
```

步骤2 在宿主机上执行如下命令，基于已构建的MindStudio镜像运行一个容器。（下面命令中的变量已在**步骤1**中设置）

```
docker run -itd \
    -v $HOME/.Xauthority:/root/.Xauthority \
    -e DISPLAY=$DISPLAY \
    -p ${port}:22 \
    --name ${container_name} \
    ${image_name}
```

表 2-13 参数解释

参数	参数说明
-itd	-i：以交互模式运行容器 -t：为容器重新分配一个伪输入终端 -d：后台运行容器，并返回容器ID
-v 宿主机目录:容器目录	将宿主机上的目录挂载到容器中。
-e DISPLAY=$DISPLAY	设置容器中的DISPLAY环境变量为宿主机上的DISPLAY环境变量。
-p *${port}*:22	将容器SSH端口22映射到宿主机的某个端口，宿主机上的端口不能被占用，可执行“netstat -anp |grep 端口号 ”，如果没有回显，表示端口未被占用
--name *${container_name}*	为容器指定一个名称
${image_name}	运行容器使用的镜像。格式为“镜像名称:镜像标签”，请根据实际情况修改

执行该命令后，如果显示容器ID，则表示mindstudio容器已经运行。

步骤3 同**2.4.4.1 在昇腾AI设备上启动MindStudio容器**的步骤3及后续步骤。

----结束

2.4.5 删除 MindStudio 容器

步骤1 停止正在运行的MindStudio容器。

```
docker stop ${container_name}
```

步骤2 删除MindStudio容器。

```
docker rm ${container_name}
```

步骤3 （可选）删除MindStudio镜像。

```
docker rmi ${image_name}
```

----结束

3 算子&模型速查

算子&模型速查工具帮助客户了解当前版本ADK支持的算子和模型。

3.1 查询算子

3.2 查询模型

3.1 查询算子

查询入口

可以通过如下两种方式进入算子查询界面：

- 在菜单栏选择“Ascend > View Supported Operators”。
- 在菜单栏单击图标。

说明

如果当前未安装有效的Ascend-Toolkit-xxx.run软件包，会弹出报错信息“No ADK is found. Please (re)install an ADK in ADK Manager first”。如果出现此报错信息，请检查Ascend-Toolkit-xxx.run软件包是否正确安装。

算子查询界面

选择“View Supported Operators”或单击图标后，弹出“Supported Operators”界面（如**图3-1**所示）。

- 用户可在“Search by Name”输入需要查询的算子名称，或根据算子的类型或硬件型号筛选需要查询的算子。其中包含算子的以下信息：
  - 算子类型（Operator Type）
  - 算子支持的硬件型号（Hardware Type）

图 3-1 Supported Operators

- 选中某个算子类型，可在详情页面查看算子具体信息，示例如图3-2。

图 3-2 算子详情示例

3.2 查询模型

查询入口

可以通过如下两种方式进入模型查询界面：

- 在菜单栏选择“Ascend > View Supported Models”。

- 在菜单栏单击图标 。

模型查询界面

支持的模型信息是从昇腾开发者社区的**ModelZoo**获取。

如果用户环境可以联网，成功获取后台信息，则弹出“Supported Models”界面，显示预训练好的模型。该界面包括模型的名称、应用领域及模型精度，如**图3-3**所示。

图 3-3 Supported Models

Search by Name:

Model Name	Application Area	Precision
ATC YOLOv3 from TensorFlow-Ascend 310	Object Detection	FP16
ATC Xception from Tensorflow - Ascend 310	Image Classification	FP16
ATC VGG19 from Tensorflow - Ascend 310	Image Classification	FP16
ATC VGG16 from Tensorflow - Ascend 310	Image Classification	FP16
ATC SqueezeNet from Tensorflow - Ascend 310	Image Classification	FP16
ATC ResNet V1 50 from Tensorflow-Ascend 310	Image Classification	FP16
ATC ResNet-101 from Tensorflow-Ascend 310	Image Classification	FP16
ATC ResNet-50v1.5 from Tensorflow-Ascend 310	Image Classification	FP16
ATC MobileNetV2 from Tensorflow - Ascend 310	Image Classification	FP16
ATC MobileNetV1 from Tensorflow - Ascend 310	Image Classification	FP16
ATC InceptionV4 from Tensorflow - Ascend 310	Classification	FP16
ATC InceptionV3 from Tensorflow - Ascend 310	Classification	FP16
ATC GoogLeNet from Tensorflow-Ascend 310	Image Classification	FP16
ATC EfficientNet-B1 from Tensorflow-Ascend 310	Image Classification	FP16
ATC DenseNet from Tensorflow-Ascend 310	Image Classification	FP16
ATC DeepFM from Tensorflow - Ascend 310	Recommendation	FP16
ATC BERT-Large (Chinese) from Tensorflow-Ascend 310	Sentiment Analysis	FP16

如果获取后台信息失败，则弹出Model Zoo对话框，显示ModelZoo的URL，如**图3-4**所示。用户可直接访问该URL查询支持的模型。

图 3-4 ModelZoo

版权所有 © 华为技术有限公司

4 模型转换

4.1 使用前必读
4.2 模型转换
4.3 模型可视化

4.1 使用前必读

用户使用Caffe/TensorFlow等框架训练好的模型，可通过ATC工具将其转换为昇腾AI处理器支持的离线模型，模型转换过程中可以实现算子调度的优化、权重数据重排、内存使用优化等，可以脱离设备完成模型的预处理，详细架构如图4-1所示。

图 4-1 ATC 工具功能架构

版权所有 © 华为技术有限公司

参考手册

表 4-1 参考手册

产品	手册
Atlas 200 DK	• 《ATC工具使用指南》 • 《应用软件开发指南（C&C++）》
其他产品	• 《ATC工具使用指南》 • 《应用软件开发指南（C&C++）》

约束说明

在进行模型转换前，请务必查看如下约束要求：

- 如果要将FasterRCNN、YoloV3、YoloV2等网络模型转成适配昇腾AI处理器的离线模型，则务必参见**15.4.1 定制网络修改（Caffe）**先修改prototxt模型文件。
- 支持原始框架类型为Caffe、TensorFlow、MindSpore、ONNX的模型转换，当原始框架类型为Caffe、MindSpore、ONNX时，输入数据类型为FP32、FP16（通过设置入参--input_fp16_nodes实现，MindSpore框架不支持该参数）、UINT8（通过配置数据预处理实现）；当原始框架类型为TensorFlow时，输入数据类型为FP16、FP32、UINT8、INT32、INT64、BOOL（原始框架类型为TensorFlow时，不支持输入输出数据类型为INT64，需要用户自行将INT64的数据类型修改为INT32类型）。
- 当原始框架类型为Caffe时，模型文件（.prototxt）和权重文件（.caffemodel）的op name、op type必须保持名称一致（包括大小写）。
- 当原始框架类型为TensorFlow时，只支持FrozenGraphDef格式。
- 不支持动态shape的输入，例如：NHWC输入为[?,?,?,3]多个维度可任意指定数值。模型转换时需指定固定数值。
- 对于Caffe框架网络模型：输入数据最大支持四维，转维算子（reshape、expanddim等）不能输出五维。
- 模型中的所有层算子除const算子外，输入和输出需要满足dim!=0。
- 只支持**15.3.1 Caffe框架算子规格**、**15.3.2 TensorFlow框架算子规格**中的算子，并需满足算子限制条件。

4.2 模型转换

模型转换入口

- 可以通过如下两种方式进入模型转换界面。
  - 在菜单栏选择“Ascend > Model Converter”。
  - 在菜单栏选择“View > Appearance > Toolbar”，菜单栏下方会出现一行工具栏，选择。

  转换详情请参见**转换步骤**。

- 用户也可以通过如下方式使用atc命令行进行模型转换：
  a. 单击MindStudio界面下方的Terminal窗口。
  b. 参见参考手册中《ATC工具使用指南》的“参数说明”章节了解相关参数作用，然后参见“使用入门”章节，设置环境变量。
  c. 参见“使用入门”章节，执行atc命令进行模型转换。

转换前提

进行模型转换前，使用MindStudio安装用户，将所转换模型的模型文件以及权重文件上传到MindStudio安装服务器。

转换步骤

步骤1 打开模型转换页面，在“Model Information”页签中上传模型文件和权重文件，如图4-2所示。

图 4-2 配置模型信息

参数解释如表4-2所示。

表 4-2 Model Information 界面参数配置

参数	说明
Model File	模型文件。必填。 有两种选择方式： ● 单击右侧的文件夹图标，在后台服务器路径选择需要转化的模型文件并上传。 ● 在参数后面的输入框中自行输入模型文件在后台服务器的路径，包括模型文件名称后缀。
Weight File	权重文件。当原始框架是Caffe时，该参数存在且必填： ● 如果模型文件和权重文件存在于后台服务器同一目录下，且名称和模型文件名称相同，则选择模型文件后，权重文件会自动填充。 ● 如果模型文件和权重文件存在于后台服务器不同目录下，或者在同一目录下，但名称和模型文件名称不相同： – 用户手动上传权重文件，单击右侧的文件夹图标，在后台服务器路径选择模型文件对应的权重文件，上传即可。 – 在参数后面的输入框中自行输入*.caffemodel权重文件在后台服务器的路径，包括权重文件名称及后缀。
Model Name	模型文件名称，必填。选择模型文件后，该参数会自动填充，用户可以根据需要自行修改名称，要求如下： 只支持a-z、A-Z、0-9、下划线以及短划线的组合，最多支持64个字符。 如果模型转换的输出路径已经存在相同名称模型文件，单击“Next”后会提示覆盖原有文件或重命名当前Model Name的信息，用户根据实际情况选择。
Target SoC Version	模型转换时指定芯片版本。请根据板端环境具体芯片形态进行选择。
Input Format	输入数据格式。 ● 当原始框架是Caffe时，取值为NCHW、ND（表示支持任意格式，N<=4），默认为NCHW。 ● 当原始框架是MindSpore、ONNX时，取值为NCHW。 ● 当原始框架是TensorFlow时，取值为NCHW、NHWC、ND、NCDHW、NDHWC，默认为NHWC。

参数	说明
Input Nodes	模型输入节点信息。 如果选择模型文件并且解析成功，则该参数下方会展示模型输入节点的shape信息以及指定输入节点类型信息。如果选择模型文件后，无法解析“Input Nodes”，该场景下，需要用户根据模型文件中的相关信息手动输入：单击该参数右侧的+，在弹出界面中输入模型输入节点的Name、Shape信息（只支持英文逗号，数字（-1或1~9））和输入节点的数据类型Type。单击删除节点信息。 如果模型有多个输入，解析成功后，“Input Nodes”参数下方会展示每一个输入节点的Shape和Type信息，其中： ● **Shape**：模型输入的shape信息，例如图4-2中的数值分别代表输入数据的N（模型一次处理的图片个数），C（Channel，例如彩色RGB图像的Channel数为3），H（Height），W（Width）。若开启AIPP功能，则此处的H，W取值即为AIPP输出数据的高和宽。 – 设置动态batch：适用于执行推理时，每次处理图片数量不固定的场景。 将解析的shape中的N设置为-1，单击其他参数或者单击“Next”，会在shape下方出现**Dynamic Batch**参数。在其中的编辑框中输入具体的档位数，每一档通过英文逗号分隔。最多支持100档配置，每个档位数值建议限制为：[1~2048]，例如输入1,2,4,8。 如果模型转换时设置了**Dynamic Batch**参数，则使用应用工程进行模型推理时，需要在**aclmdlExecute**接口之前，增加**aclmdlSetDynamicBatchSize**接口，用于设置真实的batch档位。关于**aclmdlSetDynamicBatchSize**接口的具体使用方法，请参见表4-1中的《应用软件开发指南（C&C++）》手册的“AscendCL API参考”中的**模型加载与执行**章节。 – 设置输入图片的动态分辨率：适用于执行推理时，每次处理图片宽和高不固定的场景。 将解析的shape中的H和W设置为-1，单击其他参数或者单击“Next”，会在shape下方出现**Dynamic Image Size**参数。在其中的编辑框中输入具体的动态分辨率参数，每一组参数通过英文分号分隔，组内参数使用英文逗号分隔。最多支持100档配置，例如输入112,112;224,224。 如果模型转换时设置了**Dynamic Image Size**参数，则使用应用工程进行模型推理时，需要在**aclmdlExecute**接口之前，增加**aclmdlSetDynamicHWSize**接口，用于设置真实的分辨率。关于**aclmdlSetDynamicHWSize**接口的具体使用方法，请参见表4-1中的《应用软件开发指南（C&C++）》手册的“AscendCL API参考”中的**模型加载与执行**章节。 如果模型转换时设置了动态分辨率，并且要使用“Data Pre-Processing”中的数据预处理功能，该场景下不能设置数据预处理的Crop和Padding功能。

参数	说明
	设置动态batch和动态分辨率不能同时使用，一次只能设置其中一个参数。 ● **Type**：指定输入节点的数据类型： 若原始框架类型为Caffe，支持的数据类型为FP32、FP16、Uint8。 若原始框架类型为MindSpore，支持的数据类型为FP32、Uint8。 若原始框架类型为ONNX，支持的数据类型为FP32、FP16、Uint8。 若原始框架类型为TensorFlow，支持的输入数据类型为FP32、FP16、Uint8、Int32、Int64、Bool。 只有Type取值为Uint8，**步骤2**中“Data Pre-Processing”页签（配置AIPP功能）才可以配置，其他类型不支持配置；如果模型有多个输入，只有Type取值为Uint8的节点，才可以配置“Data Pre-Processing”页签；如果Type取值为Uint8的节点无法获取Shape中的H，W信息，也无法配置“Data Pre-Processing”页签。 **说明** 若原始框架为MindSpore，Input Nodes不会自动解析对应模型中的输入信息，需要用户自行查看相应的网络模型输入，手动填写。如果不填，后台使用atc命令进行转换时，ATC工具会自动解析网络模型中的相关参数。

参数	说明
Output Nodes	指定输出节点信息。 单击“Select”在弹出的网络拓扑结构中，选中某层节点，右击选择“Select”，该层变成蓝色，单击“OK”后，在“Output Nodes”参数下面会看到标记层的算子，右击选择“Deselect”取消选中。 ● Op Name：标记层的算子名称。 ● Data Type：算子输出的数据类型，包括FP32、UINT8、FP16，通过该参数用户可以设置单个算子的输出数据类型。 “Output Nodes”参数下方“Select”层的算子，默认为全部选中，用户可以自行选择将不需要输出的算子去勾选，只有选中的算子才会作为模型的输出。本章节以选中所有算子为例进行说明。 某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以通过单击“Select”按钮，在弹出网络拓扑结构中将所需层的算子标记为“Select”，然后在“Output Nodes”参数下方选中想要输出的算子，模型转换后，在相应.om模型文件可以看到该算子的输出直接作为模型的输出。详细信息请参见**4.3 模型可视化**。 **说明** ● 如果不标记某层算子或标记后Output Nodes节点下方不选中某个算子，则模型的输出默认为最后一层的算子信息。 ● 如果标记某层算子，并且标记后Output Nodes节点下方选中某个或多个算子，则模型输出为Output Nodes节点下方选中的算子。 ● 如果模型转换过程中该算子被融合掉，则该算子不能作为输出节点。 ● 如果用户选择的模型中包括不支持的算子，则单击“Select”后，MindStudio界面“Output”窗口会提示哪些算子不支持并提示shape信息无法获取等信息，在弹出的网络结构图中该算子呈现红色，该场景下无法获取算子的输出维度和shape信息。 ● 若原始框架为MindSpore，该参数无法编辑，不支持指定输出节点信息。
Load Configuration	导入上次模型转换的配置文件。 如果用户之前转换过模型，无论成功与否，在$HOME/modelzoo/${Model Name}/device/路径都会生成${Model Name}_config.json配置文件，该文件记录用户模型转换时所选择的配置信息，包括所用模型路径、模型名称、输入输出配置，数据预处理配置等，下次重新转换模型时，通过单击“Load Configuration”选择相应路径下的配置文件，则相应的配置信息会自动填充，用户自行决定是否沿用上次配置还是修改配置后重新进行模型转换。

单击右侧的 按钮，弹出生成模型网络结构图进度条，之后会弹出该模型的原始网络结构图，如**图4-3**所示。（MindSpore框架的原始网络模型，不支持查看网络结构图。）

说明

上/下滑动鼠标滚轮，可以控制模型可视化区域的上/下移动。

按住Ctrl键+上下滑动鼠标滚轮，可以实现模型可视化区域的放大/缩小。

图 4-3 模型网络结构

- **查看算子信息**

  单击某层算子，该层算子会出现绿色选中框，右上角区域会展示该选中框所有的算子，单击 ^ 可以展开所有算子信息，包括算子名称，算子类型以及算子的参数信息等，单击 ˇ 折叠所有的算子信息；用户也可以逐层单击相关算子左侧的展开按钮 > ，查看参数的详细信息。展示的信息同模型文件中该层算子的信息。

- **查看算子输出维度和shape信息**

  图4-3所示界面展示了每一层算子输出的维度和shape信息，如每一层算子连接线中间的1,3,224,224、1,64,112,112等信息。将鼠标移至某个shape上面，在其上面会显示算子输出的维度信息，例如NCHW等维度。

  如果用户选择的模型中包括不支持的算子，则单击右侧的 按钮，MindStudio界面“Output”窗口会提示哪些算子不支持并提示shape信息无法获取等信息，在弹出的网络结构图中该算子呈现红色，该场景下无法获取算子的输出维度和shape信息。该场景下的处理方法请参见异常处理。

- **搜索算子**

  在图4-3单击右侧“Op Search”，在界面右下方会弹出搜索区域，如图4-4所示。

  搜索区域中给出了该模型所用到的所有算子，您可以在搜索区域对话框中输入算子名称，下方搜索区域会列出相关的算子，选择其中一个算子，左侧网络拓扑结

构中相应算子会显示绿色选中框，右上方会展示该选中框所包括的所有算子信息。

图 4-4 模型可视化界面的搜索功能

- **搜索算子内部信息**

  选中某层算子，单击右上方的展开按钮，右侧区域框会展示该层所包括所有算子的信息，然后单击或在右侧区域使用“Ctrl+F”快捷键，弹出图4-5所示搜索界面，其中：

  – Case sensitive：大小写敏感。
  – Wrap Search：循环搜索。
  – Backward：反向搜索。

  用户在输入框中输入想要查询的信息，比如算子的输入输出、属性等信息，然后在图4-5界面中勾选搜索特性，单击“Find”进行查找，如果匹配到相关信息，则右侧区域框中相关信息会高亮显示，否则会提示“Value not found”。单击“Close”退出搜索。

图 4-5 搜索算子内部信息

单击“OK”退出编辑。

步骤2 单击“Next”，进入“Data Pre-Processing”配置数据预处理页签，如图4-6所示。

数据预处理是昇腾AI处理器提供的硬件图像预处理模块，包括色域转换，图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等功能。

只有当“Model Information”页签，“Input Nodes”参数中，输入节点的“Type”有配置为“Uint8”类型，“Data Pre-Processing”页签才可以配置该节点的数据预处理功能。如果模型有多个输入，每个输入节点都可以获取shape信息中的宽和高，并且“Input Nodes”参数中每个输入节点的“Type”都配置为“Uint8”，则“Data Pre-Processing”页签可以配置多个节点的数据预处理功能。

图 4-6 配置数据预处理功能

参数说明如表4-3所示。

表 4-3 节点配置参数

参数	说明
Image Pre-processing Mode	图片预处理模式， 包括如下两种： ● Static：静态AIPP。 ● Dynamic：动态AIPP。
Image Pre-processing Mode取值为Static，如下参数需要配置：	

参数	说明
Input Node：(data)	节点配置开关，可以控制是否对该节点开启AIPP。只有**图4-2**中Input Nodes参数对应data节点的Type取值为UINT8，该参数才会自动开启。当模型为TensorFlow时，**图4-2**中Input Nodes参数对应data节点的Type取值为FP32、Int32、Int64、Bool时，可以手动开启该参数。
Input Node：(*im_info*)	模型有两个输入时，该参数才出现，表示对模型的第二个输入开启AIPP。只有**图4-2**中Input Nodes参数对应*im_info*节点的Type取值为UINT8，并且能获取模型的宽和高，该参数才打开。Input Node：（*im_info*)中的*im_info*根据解析的模型不同而变化。

参数	说明

Input Image Format	输入图片格式。包括如下选项：
	YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、YUV400、RGB package、BGR package、ARGB package、RGBA package、ABGR package、BGRA package。（模型转换完毕后，在对应的*.om模型文件中，上述输入格式参数分别以1、1、9、9、4、3、3、7、7、7、7枚举值呈现）
	● 若取值为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp，则右侧会出现输入数据类型选项：BT.601(Video Range)、BT.601(Full Range)、BT.709(Video Range)、BT.709(Full Range)
	不同的类型，对应不同的色域转换配置（色域转换，用于将输入的数据格式，转换为模型需要的格式，具体色域转换系数会在模型转换完成后生成在insert_op.cfg配置文件中）其中：
	– BT.601是标准清晰度视频的格式，定义于SDTV标准中。
	– BT.709是标准化高清晰度视频的格式，定义于HDTV标准中。
	两种标准又分为NARROW(Video Range)和WIDE(Full Range)，其中：
	NARROW取值范围为：Ranges: Y [16 ... 235] Cb/Cr [16 ... 240] R/G/B [0 ... 255]；WIDE取值范围为：Ranges: Y/Cb/Cr [0 ... 255] R/G/B [0 ... 255]，关于如何判断输入数据的标准，请参见**15.5 使用AIPP色域转换模型时如何判断视频流的格式标准**。
	● 若取值为YUV400，不支持色域转换。
	● 若取值为RGB package、BGR package、ARGB package、RGBA package、ABGR package、BGRA package，则不会出现右侧的输入数据类型选项，根据输出（Model Image Format）参数的取值不同，模型转换完成后，数据预处理配置文件insert_op.cfg中如下参数的取值不同：
	– 若取值为BGR package，输出为RGB；或取值为RGB package，输出为BGR：
	`#色域转换前，R通道与B通道交换开关/U通道与V通道交换开关`
	`rbuv_swap_switch : true`
	– 若取值为BGR package，输出为BGR；或取值为RGB package，输出为RGB：
	`#色域转换前，R通道与B通道交换开关/U通道与V通道交换开关`
	`rbuv_swap_switch : false`
	– 若取值为ARGB package，输出为RGBA；或取值为ABGR package，输出为BGRA：
	`#色域转换前，R通道与B通道交换开关/U通道与V通道交换开关`
	`rbuv_swap_switch : false`
	`#色域转换前，RGBA->ARGB, YUVA->AYUV交换开关`
	`ax_swap_switch : true`
	– 若取值为ARGB package，输出为BGRA；或取值为ABGR package，输出为RGBA：

参数	说明
	rbuv_swap_switch : true ax_swap_switch : true – 若取值为RGBA package，输出为RGBA；或取值为BGRA package，输出为BGRA： rbuv_swap_switch : false ax_swap_switch : false – 若取值为RGBA package，输出为BGRA；或取值为BGRA package，输出为RGBA： rbuv_swap_switch : true ax_swap_switch : false
Input Image Resolution	原始图片大小。 如果“Input Image Format”取值为“YUV420 sp”，要求原始图片的宽和高取值是偶数。
Model Image Format	模型处理图片格式。 输入图片格式不同，模型处理图片格式不同，具体格式如下： ● 若“Input Image Format”取值为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、RGB package，则“Model Image Format”取值为RGB、BGR。 ● 若“Input Image Format”取值为BGR package，则“Model Image Format”取值为RGB、BGR、GRAY。 ● 若“Input Image Format”取值为YUV400，则Model Image Format取值只能为GRAY。 ● 若“Input Image Format”取值为ARGB package、RGBA package、ABGR package、BGRA package，则“Model Image Format”取值为RGBA、BGRA。 该参数同时也是色域转换开关，默认开启。当输入图片格式与模型处理文件格式不一致时需要开启。

参数	说明
Crop	抠图开关，若开启，则表示启用抠图功能。默认关闭。 开启该参数后，下方会出现如下两个参数： ● Cropping Start：抠图开始位置。Cropping Start [H][W]的取值范围要小于Input Image Resolution [H][W]（原始图片的高和宽）。 ● Cropping Area：抠图大小，默认抠图大小的宽和高来自图4-2“Input Nodes”参数中Shape参数取值的宽和高，修改的宽和高取值不能超过原始图片Input Image Resolution对应参数的取值。 抠图约束如下： ● 若“Input Image Format”取值为YUV420 sp、YVU420 sp，则Cropping Start [H][W]都必须为偶数。 ● 若“Input Image Format”取值为YUV422 sp、YVU422 sp，则Cropping Start [W]必须为偶数。 ● 若“Input Image Format”取值为其他值，对Cropping Start [H][W]没有约束。 ● 若开启抠图功能，则Input Image Resolution >= Cropping Area + Cropping Start。 若开启抠图功能，并且没有开启Padding，该场景下Cropping Area [H][W]才能取值为0或不配置，此时抠图大小Cropping Area [H][W]取值来自图4-2Input Nodes中shape取值的高和宽（模型文件input shape中的高和宽）。
Padding	Padding使能开关，若开启，则表示启用补边功能。默认关闭。 Padding Area [L][R][B][T]取值范围为[0,32]。AIPP经过Padding后，输出的图片的高和宽要与模型需要的高和宽保持一致。
Normalization	归一化开关。 开启后，包括Mean、Min、Variance三个配置项，其中“Conversion Type”表示计算规则。
Mean	每个通道的均值。 当开启“Normalization”参数时才呈现。 ● 若“Input Image Format”为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、YUV400、RGB package、BGR package，则该参数显示为Mean: [R][G][B]，每个通道的默认值为104、117、123。 ● 若“Input Image Format”为ARGB package、RGBA package、ABGR package、BGRA package，则该参数显示为Mean: [R][G][B][A]，每个通道的默认值为104、117、123、0。

参数	说明
Min	每个通道的最小值。 当开启“Normalization”参数时才呈现。 ● 若“Input Image Format”为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、YUV400、RGB package、BGR package，则该参数显示为Min: [R][G][B]，每个通道的默认值都为0。 ● 若“Input Image Format”为ARGB package、RGBA package、ABGR package、BGRA package，则该参数显示为Min: [R][G][B][A]，每个通道的默认值都为0。
Variance	每个通道的方差。 当开启“Normalization”参数时才呈现。 ● 若“Input Image Format”为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、YUV400、RGB package、BGR package，则该参数显示为Variance: [R][G][B]，每个通道的默认值都为1.0。 ● 若“Input Image Format”为ARGB package、RGBA package、ABGR package、BGRA package，则该参数显示为Variance: [R][G][B][A]，每个通道的默认值都为1.0。
Image Pre-processing Mode取值为Dynamic，如下参数需要配置：	
Input Node：(data)	动态AIPP节点配置开关，可以控制是否对该节点开启动态AIPP。只有图4-2中Input Nodes参数对应data节点的Type取值为UINT8，该参数才打开。
Input Node：(*im_info*)	模型有两个输入时，该参数才出现，表示对模型的第二个输入开启动态AIPP。只有图4-2中Input Nodes参数对应*im_info*节点的Type取值为UINT8，并且能获取模型的宽和高，该参数才打开。Input Node：（*im_info*)中的*im_info*根据解析的模型不同而变化。
Max Image Size (Byte)	输入图像最大的size，动态AIPP必须配置（如果为动态batch场景，N为最大档位数的取值） ● 若输入图像格式为YUV400_U8，则Max Image Size>=N * Input Image Resolution [W] * Input Image Resolution [H] * 1。 ● 若输入图像格式为YUV420SP_U8，则Max Image Size>=N * Input Image Resolution [W] * Input Image Resolution [H] * 1.5。 ● 若输入图像格式为XRGB8888_U8，则Max Image Size>=N * Input Image Resolution [W] * Input Image Resolution [H] * 4。 ● 若输入图像格式为RGB888_U8，则Max Image Size>=N * Input Image Resolution [W] * Input Image Resolution [H] * 3。

须知

模型转换是否开启AIPP功能，执行推理业务时，对输入图片数据的要求：

模型转换时开启AIPP，在进行推理业务时，输入图片数据要求为NHWC排布，该场景下最终与AIPP连接的输入节点的格式被强制改成NHWC，该场景下可能与**步骤1**中“Model Information”页签中“Input Format”参数指定的数据格式不一致。

步骤3 单击“Next”，进入“Advanced Options Preview”高级选项配置页签。如下图所示。

图 4-7 高级选项配置

参数解释如下：

表 4-4 Choose Model 界面参数配置

参数	说明
Operator Fusion	是否关闭融合功能。 ● 打开表示关闭融合功能。 ● 关闭表示开启融合功能。参数默认关闭，即默认打开融合功能。 如果使用昇腾模型压缩工具量化后的模型通过模型转换得到.om离线模型，然后进行精度比对，则需要打开该开关。打开后，模型转换完毕，在生成om模型的同级目录下，会生成fusion_switch.cfg配置文件，该文件记录哪些功能被关闭。

参数	说明
Auto Tune Mode	设置算子的自动调优模式。使用该功能时，请确保MindStudio和运行环境在同一台服务器。 控制TBE算子编译时，是否对算子进行调优，以便在昇腾AI处理器上寻找最好的性能配置，包括如下几种模式： ● Genetic Algorithm（GA）：遗传算法，用于设置Cube算子的调优性能。 ● Reinforcement Learning（RL）：强化学习，用于设置Vector算子的调优性能。 ● Genetic Algorithm and Reinforcement Learning：GA和RL调优功能同时开启。 其中： ● GA调优： – ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/opp/data/tiling/${soc_version}/built-in/**路径会生成内置知识库以及cost model。该路径负责存储常用shape调优后的知识库和cost model，如果网络模型中某个shape没有命中内置知识库，则会发起调优。 – ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/tiling/${soc_version}/custom/**路径用于存放调优过程中生成的知识库。如果该路径已经有知识库，则对该知识库进行追加；如果当前路径没有知识库，则新建知识库。 进行模型转换时，只有设置Auto Tune Mode=GA或Auto Tune Mode="RL,GA"，模型编译时才会默认读取自定义知识库中的调优策略，否则模型编译时直接使用Auto Schedule进行调优。 ● RL调优： – ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/opp/data/rl/${soc_version}/built-in/**路径会生成内置知识库，该路径负责存储常用shape调优后的知识库。 – ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/rl/${soc_version}/custom/**路径用于存放调优过程中调优性能优于built-in或built-in中没有的知识库。

参数	说明
	说明 ● 如果custom路径已经有用户调优过的知识库，由于某种原因导致算子逻辑变更，例如GEMM算子新增支持的ND输入，该情况下需要在MindStudio安装服务器设置如下环境变量后，重新发起调优。 export REPEAT_TUNE=True ● 开启auto_tune时，由于上板评估需要申请内存，内存占用相比ATC工具要大，额外的内存占用大小与同时使用的device设备数有关，可近似估算为：2*device数量*输入数据大小。如果内存超过ATC运行内存大小，则会导致调优失败。 ● 使用auto_tune工具进行GA模式调优时，需要独占Device资源，不可以进行需要使用Device资源的其他操作，请将其他进程停止后，重新进行调优。如果Host有编译业务执行，则会影响调优时长。
Additional Arguments	扩展转换参数。模型转换界面不支持配置，但是ATC工具支持的参数，均可通过此选项进行扩展。最多支持输入2048个字符。 在下方编辑框中输入ATC工具可用的参数，用户根据实际情况进行填写，多个参数使用空格分隔。详细参数请参见《ATC工具使用指南》> 参数说明章节，例如：--log=info。 如果界面已经有模型转换支持的参数，例如该页签中的设置算子的自动调优模式参数“Auto Tune Mode”，对应ATC工具中的“--auto_tune_mode”参数，则在“Additional Arguments”再次配置“--auto_tune_mode”为其他数值，例如配置为“--auto_tune_mode="GA,RL"”，则“Auto Tune Mode”参数指定的模式不生效。
Command Preview	模型转换使用的atc参数预览。不支持修改。 在所有页签配置完相关参数后，该区域会给出界面参数转换成atc命令的结果预览，例如“Auto Tune Mode”参数配置为“Genetic Algorithm”，则 “Command Preview”区域展示的atc命令为“--auto_tune_mode="GA"”。 如果“Additional Arguments”中再次配置界面已有的参数，例如配置为“--auto_tune_mode="GA,RL"”，则“Command Preview”区域会追加相应的参数，模型转换时，“Command Preview”区域后面的参数取值会覆盖前面已有的参数。

步骤4 单击“Finish”，开始进行模型转换。

在MindStudio界面下方，“Output”窗口会显示模型转换过程中的日志信息，如果提示“Model converted successfully”，则表示模型转换成功。“Output”窗口会显示模型转换所用的命令、所设置的环境变量、模型转换的结果、模型输出路径以及模型转换日志路径等信息。

步骤5 模型转换完毕，在服务器后台路径“$HOME/modelzoo/*resnet50*/device”下会生成用于运行环境运行的.om模型文件，以及模型转换所用的配置信息文件${modelname}_config.json。

- 如果开启数据预处理功能，在.om同级目录下，还会生成数据预处理的配置信息文件（insert_op.cfg）。
- 如果勾选关闭融合功能，在.om同级目录下，还会生成融合开关配置文件（fusion_switch.cfg），用于记录哪些功能被关闭。

模型转换的日志文件（ModelConvert.txt）所在路径为：“$HOME/modelzoo/resnet50”。

```
hisisoc@dggphicprd32833:~/modelzoo/resnet50/device$ ll
total 150636
drwxr-x--- 2 hisisoc hisisoc     4096 Mar 10 16:46 ./
drwxr-x--- 3 hisisoc hisisoc     4096 Mar 10 16:45 ../
-rw-r----- 1 hisisoc hisisoc       98 Mar 10 15:55 fusion_switch.cfg       --融合开关配置文件
-rw-r----- 1 hisisoc hisisoc      453 Mar 10 16:45 insert_op.cfg           --数据预处理配置文件
-rw-r----- 1 hisisoc hisisoc     2095 Mar 10 18:03 resnet50_config.json    --模型转换所用的配置信息文件，
下次模型转换时，可以通过选择该文件沿用上次模型转换的配置数据
-rw------- 1 hisisoc hisisoc 51581408 Mar 10 16:46 resnet50.om             --上板运行的模型文件
```

----结束

异常处理

- **问题描述**

  如果用户选择的模型文件中包括昇腾AI处理器不支持的算子，则模型转换时会弹出图4-8所示整网支持度评估报告。

  图 4-8 模型整网支持度报告

  其中左侧的“Summary”区域中：

  – All Operator：显示本次转换的模型所包括所有算子个数，包括不支持的算子个数。

    单击“All Operator”，在右侧的“Result Details”区域，会展示模型所有的算子详细信息，包括算子类型，算子名称，是否解析成功的结果；如果算子解析失败，在“Description”处还会展示解析失败的原因。

  – UnSupported Operator：本次转换的模型不支持的算子个数。下方会列出不支持算子的原因，以及每种原因下不支持的算子个数。

    单击“UnSupported Operator”，右侧会筛选出所有不支持的算子。

- **解决方法**

  a. 在图4-8所示整网支持度评估报告界面中，选中右侧“Result Details”区域“Result”为“failed”的算子，则该算子整行被选中。单击最右侧“Operation”下面的解决方法，例如“Creator Operator”，创建自定义算子工程。

     如果当前已经打开了算子工程，则会弹出图4-9提示框，可以选择在当前算子工程添加算子或新建算子工程；如果当前不存在算子工程，则会直接弹出新建算子工程界面。

     关于创建自定义算子工程的详细操作请参见8.4 工程创建。

     图 4-9 创建算子工程时提示信息

b. 根据引导完成自定义算子工程的创建。

“New Project > Ascend Operator”中的“Operator Type”自定义算子的类型，会根据模型支持度评估界面选中的算子类型自动填充。创建完成后，新建工程的默认存储路径为“$HOME/AscendProjects”。

算子工程目录结构以及主要文件如下：

```
├── .idea
├── bulid                          //编译生成的中间文件
├── cmake                          //编译相关公共文件存放目录
├── framework                      //算子插件实现文件目录
│   ├── tf_plugin                  //存放tensorflow框架的算子插件文件及编译规则文件
│   │   ├── tensorflow_add_plugin.cpp
......
......
```

c. 进行自定义算子的开发，详细信息请参见**8 自定义算子开发**。

自定义算子开发完成后，重新进行模型转换。

4.3 模型可视化

对于已经转换成功的.om模型文件，可以在MindStudio界面呈现其网络拓扑结构，并可以查看模型所使用的算子。

步骤1 依次单击菜单栏“Ascend > Model Visualizer”，或在工具栏选择 ，在**图4-10**所示界面中选择要可视化的模型，例如此处的resnet50模型文件。

图 4-10 选择要打开的模型

步骤2 依次选择“resnet50 > device”，选择**4.2 模型转换**中已经转换成功的resnet50.om模型文件，单击“OK”，弹出**图4-11**所示界面。

图 4-11 模型可视化界面

- **查看算子信息**

  图4-11展示了模型文件中所有的算子，单击某层算子，该层算子会出现绿色选中框，右侧区域会展示该算子的详细信息，包括算子名称、算子输入、输出等信息。单击 ⌃ 展开算子的所有参数信息，用户也可以逐层单击相关参数左侧的展开按钮 ▸，查看参数的详细信息；单击 ⌄ 折叠所有的算子信息。

右下角会展示该网络模型的整体结构，包括左侧展示框中的算子在整体网络结构中的位置，即图中的蓝色框选择位置。左侧展示区域下拉，右侧整体区域中的蓝区选择框随之往下移动。左侧展示框放大和缩小，右侧蓝色框随之缩小和放大。

说明

上/下滑动鼠标滚轮，可以控制模型可视化区域的上/下移动。

按住Ctrl键+上下滑动鼠标滚轮，可以实现模型可视化区域的放大/缩小。

- **查看算子输出维度和shape信息**

  图4-11还展示了每一层算子输出的维度和shape信息，如每一层算子连接线中间的1,224,224,4等shape信息，将鼠标移至某个shape上面，在其上面会显示算子输出的维度信息，例如NCHW以及NC1HWC0等维度。

- **搜索算子**

  在图4-11单击右侧区域的“Op Search”，在界面右下方会弹出搜索区域，如图4-12所示。

图 4-12 模型可视化界面的搜索功能

搜索区域中给出了该模型所用到的所有算子，您可以在搜索区域对话框中输入算子名称，下方搜索区域会列出相关的算子。选择其中一个算子，左侧网络拓扑结构中相应算子会显示绿色选中框，右上方会展示该算子的详细信息，包括算子名称、算子输入、输出等信息。

- **搜索算子内部信息**

  选中某层算子，单击右上方的展开按钮，右侧区域框会展示该层所包括所有算子的信息，单击或在右侧区域使用“Ctrl+F”快捷键，弹出图4-13所示搜索界面，其中：

  – Case sensitive：大小写敏感。
  – Wrap Search：循环搜索。
  – Backward：反向搜索。

  用户在输入框中输入想要查询的信息，比如算子的输入输出、属性等信息，然后在图4-13界面中勾选搜索特性，单击“Find”进行查找，如果匹配到相关信息，则右侧区域框中相关信息会高亮显示，否则会提示“Value not found”。单击Close退出搜索。

图 4-13 搜索算子内部信息

- **查看模型输出节点信息**

  由于模型转换时Select了名称为**res2a_branch2a**、**bn2a_branch2a**、**scale2a_branch2a**、**res2a_branch2a_relu**的算子，则在.om模型文件可以看到Select算子的输出直接作为模型的输出，如**图4-14**中标记红色的信息，用户可以单击相应的TransData算子，查看相应的输入信息。

图 4-14 Select 算子前后区别

- **查看模型输出节点的数据类型**

  如果模型转换时设置了Select节点的输出类型，对应**4.2 模型转换**章节“Model Information”页签中**Output Nodes**区域下方节点的**Data Type**类型，模型转换完毕，用户可以单击NetOutput节点，查看不同输入节点的数据类型。

  如果模型转换时只设置了“Model Information”页签中的“Output Type”参数取值，没有Select节点，则模型转换完毕，单击NetOutput节点，查看输入数据类型即可。

  dtype为DT_FLOAT，表示数据类型为FP32，dtype为DT_FLOAT16表示数据类型为FP16，dtype为DT_UINT8表示数据类型为UINT8。

图 4-15 模型输出数据类型查看

----结束

5 模型训练

5.1 概述

5.2 操作步骤

5.3 MindSpore模型训练样例参考

5.4 MindInsight训练可视化

5.5 MindSpore模型转换后离线推理

5.1 概述

MindStudio负责运行训练框架，把框架执行的脚本、数据集、参数等相关信息提交给GE并通过接口指示GE只做网络分析并输出分析结果，MindStudio对网络分析结果进行界面展示（界面方式提供），对于网络中的不支持算子，支持用户选择触发自定义算子流程，在自定义算子流程中支持自动生成算子信息库、算子原型库和算子插件。基于MindStudio的训练流程如**图5-1**所示。

图 5-1 业务流程

说明

当安装TensorFlow框架时，需确保运行环境中已安装框架插件包Ascend-cann-tfplugin_*xxx*.run，安装方式请参见《**CANN 软件安装指南**》中“安装框架插件包”章节。

5.2 操作步骤

创建训练工程

步骤1 进入训练工程创建界面，训练工程界面如**图5-2**所示。

- 首次登录MindStudio：在MindStudio欢迎界面中单击“Create new project”，进入创建工程界面。
- 非首次登录MindStudio：在顶部菜单栏中选择“File > New > Project...”，进入创建工程界面。

图 5-2 创建工程界面

步骤2 创建训练工程。

1. 在左侧导航栏选择“Ascend Training”，如图5-2所示，在右侧配置训练工程信息，配置示例如表5-1所示。

表 5-1 工程信息配置

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	MyTraining
Description	工程描述信息，自行配置。	可选配置
Framework	框架名称，Framework支持Tensorflow、Caffe、PyTorch、MindSpore四种类型。	Tensorflow
ADK Version	当前ADK的版本号。	选择当前ADK的版本号
Project Location	工程的存储路径。	保持默认

说明

当框架选择为TensorFlow时，需确保进行模型训练的运行环境（本地环境或远程环境）中已安装框架插件包Ascend-cann-tfplugin_*xxx*.run，安装方式请参见《**CANN 软件安装指南**》中“安装框架插件包”章节。

2. 单击“Next”，进入训练工程选择界面。
3. 训练工程选择界面如图5-3，选择“Empty Project”，单击“Finish”完成模板工程的创建。

图 5-3 训练工程选择界面

4. 若工作窗口已打开其他工程，会出现如图5-4所示提示。
   - 选择“This Window”，则直接在当前工作窗口打开新创建的工程。
   - 选择“New Window”，则新建一个工作窗口打开新创建的工程。

图 5-4 工程创建提示

须知

- 新建工程的默认存储路径为安装用户家目录的“AscendProjects”文件夹下。

步骤3 查看训练工程目录结构和主要文件。

```
├── .idea
├── data                    //数据集目录
├── .project                //工程信息文件，包含工程类型、工程描述、运行目标设备类型以及ADK版本
├── train.py                //训练脚本文件，为空文件，用户在这里编写训练脚本
├── MyTraining.iml
```

须知

在训练工程执行前，需手工上传数据集到训练机器，数据集目录由用户自定义。

----结束

运行配置

步骤1 配置运行参数。

1. 单击应用工程界面“Run > Edit Configurations...”或单击图5-5所示菜单，进入运行配置界面。

图 5-5 快捷方式进入运行配置界面

2. 在运行配置界面上点击“+ ”然后选择“Ascend Training”进入运行参数配置界面。
   - “Run Mode”若选择“Remote Run”，则配置界面如图5-6所示。

图 5-6 运行配置界面

   - “Run Mode”若选择“Local Run”，则配置界面如图5-7所示。

图 5-7 运行配置界面

在右侧配置训练工程运行信息，配置示例如表5-2所示。

表 5-2 训练工程运行信息

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	MyTraining
Executable	训练工程中的执行入口文件。	train.py
Run Mode	运行环境选择。选择“Remote Run”或“Local Run”，默认为“Remote Run”。	“Remote Run”
SSH Connection	远程训练服务器IP，在“Ascend > SSH Configurations”中配置。	可选配置。
Command Arguments	训练工程执行参数。	可选配置。
Environment Variables	训练工程环境变量。	可选配置。

3. 点击“OK”完成训练工程信息的创建。

步骤2 上传数据集，创建训练脚本文件以及编写执行入口文件。

1. 上传数据集至远程训练服务器中，训练脚本中数据集导入路径需与上传后的数据集路径一致。
2. 创建训练脚本执行文件及编写训练脚本。
3. 编写执行入口文件train.py。可参考如下简单示例：

```
import os
if __name__ == '__main__':
    val = os.system(' ./npu.sh npu.py')
```

其中npu.sh为用户自行创建的训练脚本执行文件，npu.py为训练代码。npu.sh脚本执行文件可参考《**TensorFlow网络模型移植&训练指南**》或《**PyTorch网络模型移植&训练指南**》中的“模型训练”章节进行编写。

4. 执行完上述步骤后训练工程目录如下（以下仅为示例）：

```
├── .idea
├── data                    //数据集目录
├── .project                //工程信息文件，包含工程类型、工程描述、运行目标设备类型以及ADK版本
├── train.py                //训练脚本文件，为空文件，用户在这里编写训练脚本
├── npu.py                  //训练脚本，用户自行创建
├── npu.sh                  //训练脚本执行文件，用户自行创建
├── MyTraining.iml
```

----结束

执行训练

步骤1 单击应用工程界面“Run >Run”或单击**图5-8**所示菜单，执行训练。训练执行过程：连接远端训练机器 -> 打包训练工程发布到远端训练机器 -> 启动训练工程执行。

图 5-8 快捷方式执行训练

步骤2 执行训练后，在应用工程界面底部“Run”窗口显示运行实时信息，如下图：

图 5-9 运行实时信息

步骤3 如果训练成功，打印训练成功信息。如果训练失败，在工程根目录下的*out/reports*下生成训练工程的整网支持度报告network_analysis_*timestamp.report*。报告内容如下：

图 5-10 整网支持度报告

----结束

5.3 MindSpore 模型训练样例参考

本节以使用MindSpore框架为例，介绍如何将昇腾社区ModelZoo上MindSpore模型训练脚本导入到MindStudio进行训练的流程。

5.3.1 准备环境与样例

环境准备

基于MindStudio进行模型训练前，您需要做好以下环境准备：

- 参见MindSpore官方网站**安装**页面进行远端运行环境的的搭建。
- 已完成MindStudio的安装，参见**2.2 安装MindStudio(物理机、msInstaller方式）**。

注意

- 昇腾社区ModelZoo上MindSpore模型训练脚本是基于安装有昇腾AI处理器的硬件环境，因此在参考官方网站搭建MindSpore运行环境时，需选择硬件平台为Ascend 910。参见MindSpore官网进行运行环境搭建完成后，**需配置Ascend 910 AI处理器配套软件包的环境变量**。
- 安装MindSpore的环境为远端环境，MindStudio部署在本地环境中，请注意区分。

样例获取

请到昇腾开发者社区ModelZoo上获取LeNet网络模型，获取链接为**LeNet模型**。直接点击下载即可。

5.3.2 基于 ResNet50 模板样例的训练步骤

以下为创建基于ResNet50模板样例的训练工程总体步骤，用户在操作过程中如需详细了解工程信息及相关弹出窗口，请参见**5.2 操作步骤**。

步骤1 新建Ascend Train工程。

Framework选择MindSpore，如**图5-11**，其余工程信息由用户自行根据实际配置。单击“Next”，进入训练工程选择界面。

图 5-11 创建工程界面

步骤2 训练工程选择界面如图5-12，选择“Sample ResNet-50”，单击“Finish”完成模板工程的创建。

图 5-12 训练工程选择界面

步骤3 选择ResNet50模板样例后工程界面如图5-13所示。

图 5-13 模板工程界面

步骤4 通过“data/”目录下的“download_cifar10.sh”脚本下载cifar10数据集。具体操作如下：

1. 进入工程目录下的“data/”目录中。

```
cd {Project Location}/MyTraining1/data
```

   其中{Project Location}为工程的存储路径，在步骤1中由用户自行创建，例如本例为“/home/mindspore/AscendProjects”，“MyTraining1”为步骤1创建的工程名。

2. 执行如下命令下载数据集。

```
chmod +x download_cifar10.sh      # 给脚本加上可执行权限
./download_cifar10.sh
```

   数据集下载完成后如图5-14所示。

图 5-14 数据集路径

说明

- cifar10数据集也可由用户自行下载，然后拷贝至本地环境的data目录下或者远端运行环境中，拷贝到远端运行环境中需注意将训练脚本中的数据集导入路径需与上传后的数据集路径一致。
- 基于脚本下载数据集要求MindStudio所在的本地环境已联网。

步骤5 配置运行参数并运行。

1. 单击训练工程界面“Run > Edit Configurations...”或单击图5-15所示菜单，进入运行配置界面。

图 5-15 快捷方式进入运行配置界面

2. 在运行配置界面上点击“+ ”然后选择“Ascend Training”进入运行参数配置界面，如图5-16所示。

图 5-16 运行配置界面

在右侧配置训练工程运行信息，配置示例如表5-3所示。

表 5-3 训练工程运行信息

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	MyTraining。
Executable	训练工程中的执行入口文件。	scripts/run_standalone_train.sh。
Run Mode	运行环境选择。	Local Run。
SSH Connection	远程训练服务器IP，在“Ascend > SSH Configurations”中配置。	本样例运行环境选择为本地，即Run mode选择为Local Run，故此项参数不显示。
Command Arguments	训练工程执行参数。	resnet50 cifar10 ../data/cifar-10-batches-bin。
Environment Variables	训练工程环境变量。	可选参数，根据实际自行配置。
MindInsight	MindInsight使能选项	Disable。
Port	数据传输端口，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。
Summary Directory	数据记录日志存储路径，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。

说明

训练过程中如果想使用MindInsight进行训练过程可视化，可参见**5.4 MindInsight训练可视化**进行配置。

3. 点击“OK”完成训练工程信息的创建。
4. 单击应用工程界面“Run >Run”或单击图5-17所示菜单，执行训练。

图 5-17 快捷方式执行训练

训练过程如图5-18所示。

图 5-18 训练过程展示

步骤6 训练完成后，会生成模型文件.ckpt，使用该模型文件进行评估，操作如下。

1. 单击训练工程界面“Run > Edit Configurations...”或单击图5-19所示菜单，进入运行配置界面。

图 5-19 快捷方式进入运行配置界面

2. 在运行配置界面上点击“+ ”然后选择“Ascend Training”进入运行参数配置界面，如图5-20所示。

图 5-20 运行配置界面

在右侧配置训练工程运行信息，配置示例如表5-4所示。

表 5-4 训练工程运行信息

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	MyTraining。
Executable	训练工程中的执行入口文件。	scripts/run_eval.sh。
Run Mode	运行环境选择。	Local Run。
SSH Connection	远程训练服务器IP，在“Ascend > SSH Configurations”中配置。	本样例运行环境选择为本地，即Run mode选择为Local Run，故此项参数不显示。
Command Arguments	训练工程执行参数。	resnet50 cifar10 ../data/cifar-10-batches-bin train/model/resnet-1-1875.ckpt。
Environment Variables	训练工程环境变量。	可选参数，根据实际自行配置。
MindInsight	MindInsight使能选项	Disable。
Port	数据传输端口，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。
Summary Directory	数据记录日志存储路径，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。

说明

训练过程中如果想使用MindInsight进行训练过程可视化，可参见**5.4 MindInsight训练可视化**进行配置。

3. 点击“OK”完成训练工程信息的创建。
4. 单击应用工程界面“Run >Run”或单击**图5-21**所示菜单，执行模型验证。

图 5-21 快捷方式执行验证

5. 评估完成后，在“*工程名*/scripts/eval”路径下生成.air文件，用于**5.5 MindSpore模型转换后离线推理**进行模型转换与推理。

----结束

5.3.3 基于 ModelZoo 训练样例的训练步骤

以下为创建基于ModelZoo训练样例的训练工程总体步骤，用户在操作过程中如需详细了解工程信息及相关弹出窗口，请参见**5.2 操作步骤**。

步骤1 新建Ascend Train工程。

Framework选择MindSpore，如图5-22，其余工程信息由用户自行根据实际配置。单击“Next”，进入训练工程选择界面。

图 5-22 创建工程界面

步骤2 训练工程选择界面如图5-23，选择“Empty Project”，单击“Finish”完成训练工程的创建。

图 5-23 训练工程选择界面

步骤3 导入MindSpore训练脚本样例。

参见样例获取获取训练脚本，解压后将lenet文件夹下的所有文件拷贝至步骤1创建的工程目录中，如本例的路径为“{Project Location}/lenet-mindspore”,其中{Project Location}为工程的存储路径，在步骤1中由用户自行创建。导入后的工程目录及文件如图5-24。

图 5-24 样例工程

步骤4 准备数据集。

按照LeNet网络模型要求准备数据集。从**http://yann.lecun.com/exdb/mnist/**下载数据集，放在data目录下的train和test子目录中，如**图5-25**所示。

图 5-25 数据集路径

说明

- train和test子目录需由用户自行创建。
- 当数据集较小的时候，可将数据集拷贝到本地环境的data目录下；当数据集较大时，数据集需上传至**远端运行环境**中，同时需将训练脚本中的数据集导入路径需与上传后的数据集路径一致。

步骤5 配置运行参数并运行。

1. 单击应用工程界面“Run > Edit Configurations...”或单击**图5-26**所示菜单，进入运行配置界面。

图 5-26 快捷方式进入运行配置界面

2. 在运行配置界面上点击“+ ”然后选择“Ascend Training”进入运行参数配置界面，如图5-27所示。

图 5-27 运行配置界面

在右侧配置训练工程运行信息，配置示例如表5-5所示。

表 5-5 训练工程运行信息

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	lenet_mindspore。
Executable	训练工程中的执行入口文件。	train.py，当前样例的训练执行脚本为train.py。
Run Mode	运行环境选择。	Remote Run。
SSH Connection	远程训练服务器IP，在“Ascend > SSH Configurations”中配置。	根据已有配置自行选择。

参数	参数说明	示例
Command Arguments	训练工程执行参数。	--data_path='./data' --device_target='Ascend' --ckpt_path='./'。
Environment Variables	训练工程环境变量。	可选参数，根据实际自行配置。
MindInsight	MindInsight使能选项	Disable。
Port	数据传输端口，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。
Summary Directory	数据记录日志存储路径，在MindInsight选择“Enable”时显示。	本样例不使能MindInsight，故此项参数不显示。

说明

训练过程中如果想使用MindInsight进行训练过程可视化，可参见**5.4 MindInsight训练可视化**进行配置。

3. 点击“OK”完成训练工程信息的创建。
4. 单击应用工程界面“Run >Run”或单击**图5-28**所示菜单，执行训练。

图 5-28 快捷方式执行训练

训练过程如**图5-29**所示。

图 5-29 评估结果展示

----结束

5.4 MindInsight 训练可视化

概述

MindInsight为MindSpore提供了简单易用的调优调试能力。在训练过程中，可以将标量、张量、图像、计算图、模型超参、训练耗时等数据记录到文件中，通过MindInsight可视化页面进行查看及分析。

安装

MindInsight组件为MindStudio的独立组件，请参考开源社区的**安装文档**，完成MindInsight组件的安装。

须知

- MindInsight与MindSpore的版本需保持一致。

安装时默认安装路径为“/usr/local/python3.7.5/bin/”用户安装时也可指定安装路径。

配置环境变量：

1. 执行如下命令打开文件系统中的~/.bashrc文件：

   vi ~/.bashrc

   在文件最后添加如下环境变量：

   ```
   export PATH=/usr/local/python3.7.5/bin:$PATH
   ```

   “/usr/local/python3.7.5/bin”为示例安装路径，请根据实际情况配置。

   :wq!保存退出。
2. 执行命令使环境变量生效。

   source ~/.bashrc
3. 验证是否成功安装。

   执行如下命令：

   mindinsight start

   如果出现下列提示，说明安装成功

   ```
   Web address: http://127.0.0.1:8080
   service start state: success
   ```
4. 重启MindStudio以启用MindInsight组件。

操作步骤

步骤1 修改训练脚本。

使用MindInsight前，需要修改训练脚本，使用 SummaryCollector 进行数据收集后将训练过程中的数据记录下来，并指定所保存的数据的位置。在训练脚本中添加如下代码：

```
# 添加如下代码
from mindspore.train.callback import SummaryCollector
```

```
# define summary
summary_dir = "summary_dir"
summary_collector = SummaryCollector(summary_dir='./summary_dir',collect_freq=32)
cb += [summary_collector]
```

此处的“summary_dir”为相对于当前训练脚本下的路径。

步骤2 进入MindInsight配置。

有以下两种方式进入MindInsight配置：

- 工具栏选择“Run > Edit Configurations...”。
- 工具栏选择“图5-30>Edit Configurations..“。

进入”Run/Debug Configurations”界面。

图 5-30 工程设置快捷方式

步骤3 设置MindInsight组件相关参数。

“Run/Debug Configurations”界面如图5-31所示，由用户自行根据实际情况配置参数。

图 5-31 参数配置界面

此处的“summary_dir”为数据记录文件相对于工程根目录下的路径。可根据实际情况填写。

表 5-6 参数说明

参数	参数说明	示例
MindInsight	MindInsight使能选项	默认为“Disable”，使能MindInsight需选择选择“Enable”。
Port	数据传输端口	示例：20123，取值范围：1~65535。

参数	参数说明	示例
Summary Directory	数据记录日志存储路径	示例：“scripts/train/summary_dir”。

其余参数可参见表5-3。

须知

- MindInsight在训练场景下，如果“Run Mode”项选择“Remote Run”模式时不支持将IP地址设置为本机IP，如果使用本机IP建议选择“Local Run”模式。

步骤4 单击“OK”完成MindInsight组件相关参数配置，将记录可视化日志信息。并在应用工程界面底部运行窗口显示运行实时信息，如图5-32。

图 5-32 运行窗口显示信息

步骤5 可视化查看训练信息。

有以下两种方式查看实时训练信息：

- 单击运行窗口内的“Web address”后显示的链接。
- 工具栏选择“Ascend > MindInsight”。

打开Web UI，可视化查看实时训练信息。

说明

查看实时训练信息需要使用浏览器，如果未安装浏览器，请用户自行安装。

----结束

5.5 MindSpore 模型转换后离线推理

使用训练出的MindSpore模型在昇腾芯片上进行推理的步骤如下。

步骤1 打开工程。

拷贝MindStudio安装目录“MindStudio/samples/inference”下的acl_resnet50工程至用户自行指定的位置，工具栏**File>Open...**打开该工程。

若出现提示“ADK or Target is not installed for the project.”的弹窗，请单击OK后重新配置ADK。

步骤2 准备数据。

您可以参照**5.3.2 基于ResNet50模板样例的训练步骤**获取ResNet-50网络的模型文件（resnet50.air），并以MindStudio安装用户将获取的文件上传至MindStudio安装服务器。

步骤3 转换模型。

在添加模型文件前，您需要将第三方模型转换为适配昇腾AI处理器的离线模型（*.om文件），模型转换具体请参见**4.2 模型转换**。

1. 在菜单栏选择“Ascend > Model Convert”进入模型转换界面。
2. 在模型转换界面的“Model Information”页面，请参照**图5-33**配置参数：

图 5-33 Model Information

   - Model File：使用**步骤2**中的模型文件（resnet50.air）所在路径，需要选择到模型文件一级。
   - Model Name：请使用默认的“resnet50”，否则在编译代码前，您还需要修改当前工程的“src/sample_process.cpp”中的omModelPath参数值。
     ```
     const char* omModelPath = "../model/resnet50.om";
     ```
   - Input Nodes：当前样例中，由于模型输入节点的数据类型为FP32，因此“Input Nodes”区域的“Type”参数只能配置为FP32（表示float32）。
3. 单击Next进入“Data Pre-Processing”页面，请不要开启预处理开关Data Pre-Processing（默认是关闭），当前样例不支持AIPP功能。继续单击Next到达“Advanced Options Preview”页面，本页不用配置，单击Finish结束转换。
4. 模型转换完毕，在MindStudio安装服务器的“MindStudio安装用户的家目录/modelzoo/resnet50/device”目录下会生成resnet50.om文件。

步骤4 添加模型文件。

1. 右击工程名，在弹出的快捷菜单中选择“Add Model”。
2. 在弹出的窗口中，选择*.om文件。

   添加*om文件有如下要求：

   - MindStudio安装用户需对*.om文件有访问权限。
   - 总共最多添加1024个*.om文件，单个*.om文件大小不超过1G，所有*.om文件的总大小小于10G。
   - *.om文件名的名称由字母数字以及.-_字符组成，文件名长度不超过64个字符。
3. 单击“OK”，完成添加模型文件。

   添加模型文件后，可按如下目录在“Project”窗口查看模型文件。

```
├── 工程名
│   ├── model
│   │   ├──*.om          //模型文件
```

步骤5 准备推理数据。

1. 以MindStudio安装用户登录MindStudio安装服务器。
2. 切换到数据文件所在目录（**步骤1**中的acl_resnet50工程文件的“data”目录），请根据实际情况替换。
3. 执行如下命令运行transfer_pic.py脚本，将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224。在data目录下生成2个*.bin文件。

```
python3.7.5 ../script/transfer_pic.py
```

   说明

   如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，则表示缺少Pillow库，请使用**pip3.7.5 install Pillow --user**命令安装Pillow库。

4. 在MindStudio工具界面，单击刷新工程目录的图标，刷新后，在工具界面的“*工程名*/data”目录下可见*.bin文件。

   图 5-34 刷新

步骤6 编译与运行，具体操作请参见**7.3.5 编译应用工程**和**7.3.6 运行应用工程**。

如果应用工程选择在远程运行，请预先在远程运行的机器上安装Ascend-cann-toolkit开发套件包，配置acllib等动态库路径，详情参考《**CANN 软件安装指南 (开发&运行场景，通过命令行方式)**》。

图5-35为运行成功示例图。

图 5-35 运行结果

```
[INFO]  start to process file:../data/1.bin
[INFO]  model execute success
[INFO]  top 1: index[0] value[0.999945]
[INFO]  top 2: index[4] value[0.000036]
[INFO]  top 3: index[5] value[0.000018]
[INFO]  top 4: index[3] value[0.000000]
[INFO]  top 5: index[2] value[0.000000]
[INFO]  output data success
[INFO]  start to process file:../data/2.bin
[INFO]  model execute success
[INFO]  top 1: index[1] value[0.999242]
[INFO]  top 2: index[9] value[0.000595]
[INFO]  top 3: index[8] value[0.000121]
[INFO]  top 4: index[6] value[0.000040]
[INFO]  top 5: index[0] value[0.000001]
```

----结束

6 脚本转换

6.1 PyTorch GPU2Ascend

6.2 MindConverter

6.1 PyTorch GPU2Ascend

6.1.1 概述

NPU是AI算力的发展趋势，但是目前训练和在线推理脚本大多还基于GPU。由于NPU与GPU的架构差异，基于GPU的训练和在线推理脚本不能直接在NPU上使用，需要转换为支持NPU的脚本后才能使用。

说明

- 脚本转换工具根据适配规则，对用户脚本给出修改建议并提供转换功能，大幅度提高了脚本迁移速度，降低了开发者的工作量。但转换结果仅供参考，仍需用户根据实际情况做少量适配。
- 脚本转换工具当前仅支持PyTorch训练脚本转换。

6.1.2 生成报告

操作步骤

步骤1 启动脚本转换。

有以下三种方式发起脚本转换任务：

- 在工具栏选择“Ascend > Framework Trans > PyTorch GPU2Ascend”。
- 右键单击训练工程，然后选择“PyTorch GPU2Ascend”。
- 单击工具栏中图标。

步骤2 参数配置。

PyTorch GPU2Ascend界面如**图6-1**所示，用户自行根据实际情况配置参数。

图 6-1 PyTorch GPU2Ascend 参数配置

表 6-1 PyTorch GPU2Ascend 参数说明

参数	参数说明
Input Path	训练工程路径。
Framework	框架类型。
Scene	使用场景。
Output Path	输出路径。

步骤3 单击“Transplant”生成结果报告。

完成后，可在下方脚本转换窗口查看解析结果报告。

说明

会在新的MindStudio窗口中打开结果报告。

----结束

报告目录

```
├── xxx_msft                    // 脚本转换结果输出目录，默认为原始脚本路径。xxx为原始脚本所在文件夹名称。
│   ├── out_reports             // 脚本转换结果分析报告目录。
│   │   ├── orig_script         // 存放原始脚本文件。
│   │   │   ├── 原始脚本文件
│   │   ├── msFmkTranspltResult.json   // 脚本转换结果分析报告文件。
│   ├── out_todo                // TODO注释结果目录，可根据注释对脚本进行修改，完成脚本转换。
│   │   ├── 加了TODO注释后的脚本文件
│   ├── msFmkTranspltlog.txt    // 脚本转换过程日志文件。
```

6.1.3 执行转换

在新的MindStudio窗口中生成的报告如图6-2所示。

图 6-2 报告

参数及按钮说明

表 6-2 参数及按钮说明

参数	参数说明
FrameWork	执行转换脚本的框架。
Total	需要进行脚本转换的总条数。
Selected	选中进行脚本转换的条数。
NO.	转换项序号。
File	执行转换代码所在文件。
Line	执行转换代码所在行数。
Content	执行转换代码原内容。
Description	具体转换描述。
☐	勾选表示执行转换应用，不勾选表示不执行转换应用。 表头处支持进行全选和全不选。
Apply	对勾选的转换项执行脚本转换。

批量转换

步骤1 勾选需要的转换项。

步骤2 单击Apply按钮。

步骤3 在弹出的确认框中单击Yes。

----结束

单个转换

步骤1 双击图6-2中要转换的项。

步骤2 单击代码编辑窗口中弹出的“Apply Auto-Convert”。

图 6-3 转换

----结束

说明

批量转换和单个转换完成后，已经转换的项会在报告窗口中消失。

6.2 MindConverter

6.2.1 概述

MindConverter是一款用于将PyTorch脚本文件、TensorFlow框架模型文件转换到MindSpore脚本的工具。结合转换报告的信息，用户只需对转换后的脚本进行微小的改动，即可快速将PyTorch脚本文件、TensorFlow框架的模型迁移到MindSpore。

6.2.2 前提条件

此工具为MindInsight的子模块，需安装MindInsight后才可使用MindConverter工具，请参考开源社区的安装文档完成MindInsight安装。

说明

1. PyTorch、TensorFlow不作为MindInsight明确声明的依赖库。若想使用基于图结构的脚本生成工具，需要用户手动安装与生成PyTorch模型版本一致的PyTorch库(MindConverter请使用PyTorch 1.4.0(TorchVision 0.5.0)进行脚本生成)，或TensorFlow1.15.0。
2. 用户在使用MindConverter时，用户除安装可满足导出的Pb模型加载、训练、推理的TensorFlow版本外，还需要将如下第三方库安装或升级至相应版本：
   onnx>=1.8.0
   tf2onnx>=1.7.1
   onnxruntime>=1.5.2
   - PyTorch模型脚本转MindSpore的用户无需安装tf2onnx。
3. 脚本转换工具本质上为算子驱动，对于MindConverter未维护的PyTorch或ONNX算子与MindSpore算子映射，将会出现相应的算子无法转换的问题，对于该类算子，用户可手动修改，或基于MindConverter实现映射关系，向MindInsight仓库贡献。

注意

模型文件的安全性与一致性请用户自行保证。

更多详情请参考**MindConverter开源社区**。

6.2.3 操作步骤

操作步骤

步骤1 启动脚本转换。

有以下两种方式发起脚本转换任务：

- 在工具栏选择“Ascend > Framework Trans>MindConverter”。
- 单击工具栏中图标。

MindConverter启动后界面如**图6-4**所示。

图 6-4 MindConverter 参数配置界面

步骤2 选择脚本转换方案。

1. 基于抽象语法树(Abstract Syntax Tree, AST)的脚本转换。
   - AST方案仅支持PyTorch模型脚本迁移，不支持TensorFlow模型脚本迁移。
   - AST方案通过对原PyTorch脚本的抽象语法树进行解析、编辑，将其替换为MindSpore的抽象语法树，再利用抽象语法树生成代码。转换后脚本保持原有PyTorch脚本结构（包括变量、函数、类命名等与原脚本保持一致）。

– 理论上，AST方案支持任意模型脚本迁移，但语法树解析操作受原脚本用户编码风格影响，可能导致同一模型的不同脚本最终的转换率存在一定差异。

AST方案如图6-5所示，由用户自行根据实际情况配置参数。

图 6-5 AST 方案参数配置

表 6-3 AST 方案参数说明

参数	参数说明	示例
Conversion Mode	转换方式	基于抽象语法树模式需选择“AST”。
Input Path	待转换PyTorch脚本路径	示例：“/home/user/AscendProject/output”，需根据实际脚本路径选择。
Output Path	转换后PyTorch脚本输出路径	示例：“/home/user/AscendProject/vgg.py”，需根据实际脚本路径选择。
Report Path	(可选) 转换报告输出路径	示例：“/home/user/AscendProject/output/report”，缺省时默认为“<Output Path>/report”。
Project Path	(可选) 待转换PyTorch脚本工程路径	缺省。

2. 基于图结构的脚本生成。
   - 计算图作为一种标准的模型描述语言，可以消除用户代码风格多样导致的脚本转换率不稳定的问题。迁移后脚本保持较高的转换率，尽量少的修改、甚至不需要修改，即可实现迁移后模型脚本的执行。
   - 在已支持算子的情况下，基于图结构的脚本生成方案可提供优于AST方案的转换率。
   - 基于图结构的脚本生成支持PyTorch、TensorFlow模型转换，只需选择相应选项并设置参数。

须知

- 基于图结构的脚本生成方案，目前仅支持单输入、单输出模型，对于多输入模型暂不支持；
- 基于图结构的脚本生成方案，由于要基于推理模式加载PyTorch模型，会导致转换后网络中Dropout算子丢失，需要用户手动补齐。

a. PyTorch模型脚本生成

   基于图结构的PyTorch模型脚本生成的参数配置如图6-6所示，由用户自行根据实际情况配置参数。

图 6-6 PyTorch 模型脚本生成参数配置

表 6-4 Pytorch 模型脚本生成的 Graph 方案参数说明

参数	参数说明	示例
Conversion Mode	转换方式	基于图结构模式需选择“Graph”。

参数	参数说明	示例
Framework	深度学习框架	默认为“Pytorch”。PyTorch模型脚本生成需选择“Pytorch”
Input Path	待转换PyTorch模型路径	示例：“/home/user/AscendProject/model/vgg16_bn.pth”，需根据实际模型路径选择。
Output Path	转换后模型输出路径	示例：“/home/user/AscendProject/output”，需根据实际脚本路径选择。
Shape	模型期望的输入样本shape	形式：“NCHW”，示例：“1,3,224,224”。
Report Path	(可选) 转换报告输出路径	示例：/home/user/AscendProject/output/report”，缺省时默认为“<Output Path>/report”。
Project Path	(可选) 待转换PyTorch脚本工程路径	缺省。

b. TensorFlow模型脚本生成

使用TensorFlow模型脚本迁移，需要先将TensorFlow模型导出为pb格式，并且获取模型输入节点、输出节点名称。使用TensorFlow导出pb模型可参考**15.8.5 TensorFlow导出Pb模型**。

基于图结构的TensorFlow模型脚本生成的参数配置如**图6-7**所示，由用户自行根据实际情况配置参数。

图 6-7 TensorFlow 模型脚本生成参数配置

表 6-5 TensorFlow 模型脚本生成的 Graph 方案参数说明

参数	参数说明	示例
Conversion Mode	转换方式	基于图结构模式需选择“Graph”。
Framework	深度学习框架	默认为“Pytorch”，TensorFlow模型脚本生成需选择“Tensorflow”。
Input Path	待转换TensorFlow模型路径	示例：“/home/user/AscendProject/model/tf_resnet50.pb”，需根据实际模型路径选择。
Output Path	转换后模型输出路径	示例：“/home/user/AscendProject/output”，需根据实际脚本路径选择。
Shape	模型期望的输入样本shape	形式：“NHWC”，示例：“1,224,224,3”。
Input Nodes	输入节点名称	示例：“input_1:0”。
Output Nodes	输出节点名称	示例：“predictions/Softmax:0”。

参数	参数说明	示例
Report Path	(可选) 转换报告输出路径	示例：/home/user/AscendProject/output/report”，缺省时默认为“<Output Path>/report”。
Project Path	(可选) 待转换TensorFlow脚本工程路径	缺省。

步骤3 单击“Transplant”进行模型脚本转换并生成转换报告。

完成后，可在下方脚本转换窗口或报告输出路径下查看转换报告。

----结束

7 应用开发

7.1 准备文档
7.2 基于MindX SDK开发应用
7.3 基于AscendCL开发应用
7.4 应用工程调试

7.1 准备文档

本章节以C/C++为例进行应用开发介绍，需要准备**表7-1**中手册作为参考。

表 7-1 参考手册

产品	手册
Atlas 200 DK	• 《**应用软件开发指南（C&C++）**》 • 《**日志参考**》
其他产品	• 《**应用软件开发指南（C&C++）**》 • 《**日志参考(推理)**》

7.2 基于 MindX SDK 开发应用

7.2.1 创建应用工程

说明

MindX SDK当前适用Ubuntu 18.04和CentOS 7.6系统，不适用于EulerOS。

步骤1 进入工程创建页面。

- 首次登录MindStudio：单击**Create New Project**。
- 非首次登录MindStudio：在顶部菜单栏中选择**File > New > Project...**。

步骤2 在**New Project**窗口中，选择**Ascend App**，配置工程相关参数如表7-2所示。

表 7-2 工程参数说明

参数	说明
Name	工程名称，自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不超过64个字符。
Description	工程描述信息，自行配置（最多256个字符）。
ADK Version	当前ADK的版本号。
Project Location	工程的存储路径，用户可自定义。

步骤3 单击**Next**，在**New Project**窗口中，选择工程类型。

- Empty MindX Project：MindX SDK空白工程。
- MindX Project Sample - Detection：MindX SDK样例工程。

步骤4 单击**Finish**，完成工程创建。

工程目录中的pipeline文件具体可参照7.2.4 可视化流程编排自行准备与编辑。MindX SDK样例工程中已经准备好pipeline文件以供用户使用，无需准备。

- 如果选择MindX SDK Project，成功创建工程后，工程目录的主要结构如下，请以实际获取为准：

```
├── 工程名
    ├── CMakeLists.txt          //编译脚本，调用src目录下的CMakeLists文件。
    ├── 工程名.iml              //IntelliJ IDEA自动创建的模块文件，用于Java应用开发，存储一些模块开发相关的信息。
    ├── build                   //存放cmake依赖文件
    ├── config
    │   └── logging.conf        //日志配置文件
    ├── out                     //存放编译出的可执行文件
    └── src
        ├── CMakeLists.txt      //编译脚本
        └── main.cpp            //主函数的实现文件，当前主函数内无代码逻辑
```

- 如果选择MindX SDK Sample - Detection，成功创建工程后，工程目录的主要结构如下，请以实际获取为准：

```
├── 工程名
    ├── CMakeLists.txt          //编译脚本，调用src目录下的CMakeLists文件。
    ├── 工程名.iml              //IntelliJ IDEA自动创建的模块文件，用于Java应用开发，存储一些模块开发相关的信息。
    ├── build                   //存放cmake依赖文件
    ├── config
    │   └── logging.conf        //日志配置文件
    ├── data
    │   └── test.jpg            //测试图片
    ├── models
    │   ├── resnet50                                  //resnet50模型相关配置文件
    │   │   ├── aipp_resnet50_224_224.aippconfig      //模型转换需要的aipp配置文件
    │   │   ├── imagenet1000_clsidx_to_labels.names   //imagenet1000索引与label对应关系
    │   │   └── resnet50_aipp_tf.cfg                  //后处理配置文件
    │   └── yolov3                                    //yolov3模型相关配置文件
    │       ├── aipp_yolov3_416_416.aippconfig        //模型转换需要的aipp配置文件
    │       ├── coco.names                            //索引与label关系
    │       └── yolov3_tf_bs1_fp16.cfg                //后处理配置文件
    ├── out                     //存放编译出的可执行文件
    ├── pipeline
```

```
│   ├── Sample.pipeline       //目标检测对应的pipeline
│   └── SamplePlugin.pipeline        //自定义插件对应的pipeline
└── src
    ├── CMakeLists.txt              //编译脚本
    └── main.cpp                   //主函数的实现文件
```

----结束

7.2.2 获取与安装 SDK 软件包

步骤1 在昇腾开发者社区的MindX SDK专区按需要获取mxManufacture软件包、mxVision软件包并上传。

步骤2 进入工程创建页面，工具栏点击**Ascend>MindX SDK Manager**进入MindX SDK管理界面。界面中MindX SDK Location为软件包的默认安装路径，非root用户为“$HOME/Ascend/mindx_sdk”，root用户为“/usr/local/Ascend/mindx_sdk”。

单击**Install SDK**进入Installation settings界面。

步骤3 在Installation settings界面，如图7-1：

- SDK：请用户选择步骤1中获取的安装包的上传路径，需要选择到软件包名一级。
- Install Location：显示软件包的默认安装路径，非root用户为“$HOME/Ascend/mindx_sdk”，root用户为“/usr/local/Ascend/mindx_sdk”。

单击**Next**后会显示选中的SDK的安装信息，图7-2为安装信息示例，继续单击**Next**开始安装。

图 7-1 Installation settings

图 7-2 Current settings

步骤4 安装完成，安装成功示例如图7-3。

图 7-3 安装完成

安装完成后单击Finish结束，返回SDK管理界面。

步骤5 在SDK管理界面，可查看安装后的SDK的信息，如图7-4所示，单击**OK**结束安装流程。

界面中两个按钮介绍如下：

- ：卸载按钮。
- ：激活按钮。

说明

对激活按钮说明如下：

- 所有安装版本的软件中，只能激活一个版本的软件使用。
- 若已有激活的某一版本的SDK软件正在使用并又安装另一版本的SDK软件，原先使用的SDK软件会被去除激活，并自动激活最新安装的SDK软件，用户如果需要继续使用原先的软件，需要手动激活。

图 7-4 安装完成后的 MindX SDK 管理界面

----结束

7.2.3 开发应用

7.2.3.1 典型业务流程

典型推理业务流程如图7-5所示，先通过目标检测获取目标坐标，然后通过图像分类识别目标属性。流程中有两次图像缩放的操作，两者主要区别在于设置的缩放宽高参数不同。

图 7-5 典型推理业务流程

7.2.3.2 开发流程

说明

- 请注意，通过MindStudio实现SDK应用开发不支持Dump功能。
- 当前MindStudio只支持C++。
- 加载的模型路径中不能有空格。

使用MindStudio开发和运行推理业务步骤如图7-6所示，其中“深入开发”为SDK高级特性，用户可以视情况选择执行。

图 7-6 推理业务开发与运行流程

步骤1 确定业务流程：

根据业务的功能如目标检测、图像分类、属性识别等，将业务流程进行模块化，请参见7.2.3.1 典型业务流程。

步骤2 寻找合适插件：

首先根据已有SDK插件的功能描述和规格限制来匹配业务功能。当SDK提供的插件无法满足功能需求时，用户可以开发自定义插件。请分别参见《mxManufacture 用户指南》或《mxVision 用户指南》中的“已有插件介绍”与“插件开发介绍”章节。

步骤3 准备推理模型文件与数据：

根据插件的实际应用场景，添加用于推理的模型文件和数据集。

1. 转换模型。

   在添加模型文件前，请参见4 模型转换将第三方模型转换为适配昇腾AI处理器的离线模型（*.om文件）。SDK样例工程请参见7.2.6 SDK样例工程使用指导中的步骤步骤3至步骤5。

2. 添加模型文件。

   请用户将准备好的模型文件上传到应用工程中用户自定义目录下。

3. 准备推理数据。

   准备推理所用数据，并上传到应用工程文件目录下。

步骤4 流程编排：

请参见7.2.4 可视化流程编排、《mxManufacture 用户指南》或《mxVision 用户指南》中的“流程编排介绍”与“数据流程介绍”章节。

步骤5 业务集成：

编写C++程序，调用业务流管理的API（MxStreamManager），先进行初始化，再加载业务流配置文件（*.pipeline），然后根据stream配置文件中的StreamName往指定Stream获取输出数据，最后销毁Stream。

可用API请参见《mxManufacture 用户指南》或《mxVision 用户指南》中的“API参考（C++）”章节。

步骤6 编译与运行应用：

请参见7.2.5 编译与运行应用工程。

----结束

7.2.4 可视化流程编排

MindX SDK实现功能的最小粒度是插件，每一个插件实现特定的功能，如图片解码、图片缩放等，将这些插件按照合理的顺序编排，实现负责的功能。可视化流程编排是以可视化的方式，开发数据流图，生成pipeline供应用框架使用。

步骤1 进入工程创建页面，在顶部菜单栏中选择**Ascend>MindX SDK Pipeline**。

步骤2 出现pipeline绘制界面，如图7-7。

图 7-7 pipeline 绘制界面

界面分为：左侧插件库、中间编辑区、右侧插件属性展示区。

- 插件库：根据已安装的MindX SDK动态扫描获取，以保证生成的pipeline可以找到对应的so。
- 编辑区：可以从插件库中拖动某个插件，放入编辑区，可以连接两个插件，形成功能数据流，支持1对1、1对多、多对1等连接关系。同时扫描出每个插件的可配置的属性信息，用户可以自定义配置。
- 插件属性展示区：点击编辑区中的插件时，会展示插件的属性。

界面按键介绍如下：

- Format：格式化按键，一键整理。
- New：新建画板。
- Open：打开pipeline文件。
- Save：保存绘制完成的插件流程图为pipeline文件。

步骤3 在左侧编辑框选择插件，拖动至中间编辑框，按照用户的业务流程进行连接。如果拖动错误插件或者错误连线，选中后键盘Del键删除。用户自定义的流水线绘制完成后，选中流水线中的所有插件，右键选择Set Stream Name设置Stream名称，如果有多条流水线则需要对每一条流水线设置Stream名称。

说明

- 设置Stream name为必选项。所有插件都需要设置Stream name，否则保存时会出现插件标红并引起报错。
- 设置Device Id为可选项。如果需要对插件设置Device Id，请将属于同一Stream的插件选中设置为相同的Device Id。

步骤4 绘制完成后，单击**Save**按钮并配置以下几项，保存pipeline文件。

- Pipeline Filename：pipeline文件的名称。
- Output Path：保存的路径。

步骤5 流程编排完成后生成pipeline配置文件。

MindStudio支持直接打开pipeline文件进行编辑。

----结束

7.2.5 编译与运行应用工程

说明

- 若SDK已有插件满足开发需求，则直接执行以下的编译运行操作。
- 若已有插件不满足开发需求，用户需要开发自定义插件并编写相应的CMakeLists文件，执行编译操作得到插件动态库文件。将后处理库文件放置于工程目录下的lib目录下，将插件库文件放置于工程目录下的“lib/plugins”目录下，流程编排可自动识别放置于该目录下的自定义插件。完成以上操作后继续执行以下的编译运行操作。
- SDK暂不支持交叉编译。

编译应用工程请参见**7.3.5 编译应用工程**。

运行应用工程请参见**7.3.6 运行应用工程**。

7.2.6 SDK 样例工程使用指导

步骤1 请参照**7.2.2 获取与安装SDK软件包**完成软件包的获取与安装。

步骤2 创建应用工程。

请参见**7.2.1 创建应用工程**完成应用工程的创建，工程类型选择MindX Project Sample - Detection。

步骤3 参见**表7-3**，前往昇腾开发者社区下载YOLOv3和ResNet50模型pb文件。

表 7-3 模型获取

模型名称	版本	获取链接
YOLOv3	1.1	获取地址 关键词搜索：YOLOv3 应用级别：Released 框架：TensorFlow 进入模型页面后，选择“历史版本”中版本1.1下载。
ResNet50		获取地址 关键词搜索：ResNet50 应用级别：Released 框架：TensorFlow 进入模型页面后，选择“历史版本”中版本1.1下载。

步骤4 模型转换。

在工程创建页面的工具栏选择**Ascend>Model Converter**。

1. 在Model Information页面，页面参数配置指导如下：

   图7-8为YOLOv3模型转换的配置参数示例，图7-9为ResNet50模型转换的配置参数示例。

   - Model File：

     获取模型pb文件后，选择上传到环境中的模型pb文件路径。需要选择到pb文件一级。

   - Model Name：

     转换YOLOv3模型时请以**yolov3_tf_bs1_fp16**命名。

     转换ResNet50模型时请以**resnet50_aipp_tf**命名。

   - Target SoC Version：

     模型转换时指定芯片版本。转换YOLOv3模型与转换ResNet50模型无差异，请根据使用的芯片类型选择。

   - Input Format：

     输入数据格式。转换YOLOv3模型与转换ResNet50模型无差异，请使用默认设置，不需要修改。

   - input/input_data：

     转换YOLOv3模型时，输入Shape请使用**1.416.416.3**、输入Type请使用**FP32**。

     转换ResNet50模型时，输入Shape请使用**1.224.224.3**、输入Type请使用**FP32**。

   - Output Nodes：指定输出节点信息。

     转换YOLOv3模型时，请选择**conv_lbbox/BiasAdd**、**conv_mbbox/BiasAdd**、**conv_sbbox/BiasAdd**三项。

     转换ResNet50模型时，无需选择Output Nodes。

图 7-8 YOLOv3 模型参数配置

图 7-9 ResNet50 模型参数配置

2. 单击Next，进入Data Pre-Processiong页面，参照图7-10作为示例配置参数。

图 7-10 Data Pre-Processiong

3. 单击**Next**，进入Advanced Options Preview页面，参照**图7-11**配置参数。

   Additional Arguments处请填写模型aipp配置文件的绝对路径，$HOME为启动MindStudio的用户家目录，*MyApp*为新建SDK样例工程名，请用户自行替换。

   转换YOLOv3模型时请配置：

   ```
   --insert_op_conf=$HOME/AscendProjects/MyApp/models/yolov3/aipp_yolov3_416_416.aippconfig
   ```

   转换ResNet50模型时请配置：

   ```
   --insert_op_conf=$HOME/AscendProjects/MyApp/models/resnet50/aipp_resnet50_224_224.aippconfig
   ```

图 7-11 Advanced Options Preview

步骤5 将转换完成的模型文件请分别放置于“*工程文件根目录*/models/resnet50”和“*工程文件根目录*/models/yolov3”下。

步骤6 编译与运行请参见**7.2.5 编译与运行应用工程**。

如果出现以下显示，表示SDK样例工程运行成功。

```
Results:{"MxpiObject":[{"MxpiClass":[{"classId":163,"className":"beagle","confidence":
0.87109375}],"classVec":[{"classId":16,"className":"dog","confidence":0.99641436299999997,"headerVec":
[]}],"x0":125.63256800000001,"x1":918.29089399999998,"y0":116.434044,"y1":597.21276899999998}]}
```

----结束

7.3 基于 AscendCL 开发应用

7.3.1 创建应用工程

步骤1 进入工程创建页面。

- 首次登录MindStudio：单击**Create New Project**。
- 非首次登录MindStudio：在顶部菜单栏中选择**File > New > Project...**。

步骤2 在**New Project**窗口中，选择**Ascend App**，配置工程相关参数如**表7-4**所示。

表 7-4 工程参数说明

参数	说明
Name	工程名称，自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不超过64个字符。
Description	工程描述信息，自行配置（最多256个字符）。
ADK Version	当前ADK的版本号。
Project Location	工程的存储路径，用户可自定义。

步骤3 单击**Next**，在**New Project**窗口中，选择工程类型。

- Empty Acl Project：创建一个仅包括开发框架的工程，不含具体的代码逻辑。
- Acl Project Sample-ResNet50：创建一个以acl_resnet50样例为模板的工程。

  acl_resnet50样例简要介绍如下：

  该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

  将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行同步推理，分别得到推理结果后，再对推理结果进行处理，输出top5置信度的类别标识。

图 7-12 Sample 示例

步骤4 单击**Finish**，完成工程创建。

- 如果选择Empty Acl Project，成功创建工程后，工程目录的主要结构如下，请以实际获取为准：

```
├── 工程名
│   ├── .idea                //IntelliJ IDEA自动创建的，用于存放项目的配置信息。
│   ├── build
│   │   ├──cmake             //存放cmake依赖文件
│   ├── out                  //存放编译出的可执行文件。
│   ├── src
│   │   ├── acl.json         //系统初始化的配置文件
│   │   ├── CMakeLists.txt   //编译脚本
│   │   ├── main.cpp         //主函数的实现文件，当前主函数内无代码逻辑
│   ├── .project             //工程信息文件，包含工程类型、工程描述、运行目标设备类型以及ADK版本。
│   ├── CMakeLists.txt       //编译脚本，调用src目录下的CMakeLists文件。
│   ├── 工程名.iml           //IntelliJ IDEA自动创建的模块文件，用于Java应用开发，存储一些模块开发相关的信息。
```

- 如果选择Acl Project Sample-ResNet50，成功创建工程后，工程目录的主要结构如下，请以实际获取为准：

```
├── 工程名
│   ├── .idea                //IntelliJ IDEA自动创建的，用于存放项目的配置信息。
│   ├── build
│   │   ├──cmake             //存放cmake依赖文件
│   ├── inc
│   │   ├── model_process.h  //声明模型处理相关函数的头文件
│   │   ├── sample_process.h //声明资源初始化/销毁相关函数的头文件
│   │   ├── utils.h          //声明公共函数（例如：文件读取函数）的头文件
│   ├── out                  //存放编译出的可执行文件
│   ├── script
```

```
│   │   ├── transferPic.py      //将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224
│   ├── scripts                 //适配CI工程的脚本目录
│   │   ├── host_version.conf
│   │   ├── testcase_200dk.sh
│   │   ├── testcase_300.sh
│   ├── src
│   │   ├── acl.json            //系统初始化的配置文件
│   │   ├── CMakeLists.txt      //编译脚本
│   │   ├── main.cpp            //主函数，图片分类功能的实现文件
│   │   ├── model_process.cpp   //模型处理相关函数的实现文件
│   │   ├── sample_process.cpp  //资源初始化/销毁相关函数的实现文件
│   │   ├── utils.cpp           //公共函数（例如：文件读取函数）的实现文件
│   ├── .project                //工程信息文件，包含工程类型、工程描述、运行目标设备类型、ADK版本号等
│   ├── CMakeLists.txt          //编译脚本，调用src目录下的CMakeLists文件
│   ├── 工程名.iml              //IntelliJ IDEA自动创建的模块文件，用于Java应用开发，存储一些模块开发相关的信息。
│   ├── readme.md               //样例工程的readme文件
```

----结束

7.3.2 准备模型文件和数据

创建 Empty Acl Project 工程时

根据实际应用的场景，添加用于推理的模型文件和图片数据集。

步骤1 转换模型。

在添加模型文件前，您需要先参见**4 模型转换**将第三方模型转换为适配昇腾AI处理器的离线模型（*.om文件）。

步骤2 添加模型文件。

1. 右击工程名，在弹出的快捷菜单中选择“Add Model”。
2. 在弹出的窗口中，选择*.om文件，可一次添加多个om文件（按住键盘上的Ctrl键选中多个文件）。

   添加*.om文件有如下要求：

   - MindStudio安装用户需对*.om文件有访问权限。
   - 总共最多添加1024个*.om文件，单个*.om文件大小不超过1G，所有*.om文件的总大小小于10G。
   - *.om文件名的名称由字母数字以及.-_字符组成，文件名长度不超过64个字符。
3. 单击“Add”，完成添加模型文件。

   添加模型文件后，可按如下目录在“Project”窗口查看模型文件。

```
├── 工程名
│   ├── model
│   │   ├──*.om          //模型文件
```

步骤3 准备推理数据。

准备推理所用数据，并上传到应用工程文件目录下。

----结束

创建 Acl Project Sample-ResNet50 工程时

步骤1 准备数据。

您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、预训练模型文件（*.caffemodel），并以MindStudio安装用户将获取的文件上传至MindStudio安装服务器。

- 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
- 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。

步骤2 转换模型。

在添加模型文件前，您需要先参见**4 模型转换**将第三方模型转换为适配昇腾AI处理器的离线模型（*.om文件）。

- 模型转换时，在模型转换页面的“Model Information”页签中：
  - 模型转换时，使用**步骤1**中的模型文件（*.prototxt）、预训练模型文件（*.caffemodel）。
  - “Model Name”参数值请使用默认的“resnet50”，否则在编译代码前，您还需要修改当前工程的“src/sample_process.cpp”中的omModelPath参数值。
    ```
    const char* omModelPath = "../model/resnet50.om";
    ```
  - 当前样例中，由于模型输入节点的数据类型为FP16，因此“Input Nodes”区域的“Type”参数只能配置为FP16（表示float16）。
- 模型转换时，在模型转换的AIPP页面，请不要开启预处理开关Data Pre-Processing（默认是关闭），当前样例不支持AIPP功能。
- 模型转换时，在“Advanced Options Preview”页面中，打开“Additional Arguments”参数处的开关，增加配置：--output_type=FP32
  - 当前样例中，由于处理推理结果时采用FP32（表示float32）数据类型，因此增加“--output_type=FP32”配置，否则处理结果不正确。
- 模型转换完毕，在MindStudio安装服务器的“MindStudio安装用户的家目录/modelzoo/*resnet50*/device”目录下会生成resnet50.om文件。

步骤3 添加模型文件。

1. 右击工程名，在弹出的快捷菜单中选择“Add Model”。
2. 在弹出的窗口中，选择*.om文件，可一次添加多个om文件（按住键盘上的Ctrl键选中多个文件）。

   添加*om文件有如下要求：
   - MindStudio安装用户需对*.om文件有访问权限。
   - 总共最多添加1024个*.om文件，单个*.om文件大小不超过1G，所有*.om文件的总大小小于10G。
   - *.om文件名的名称由字母数字以及.-_字符组成，文件名长度不超过64个字符。
3. 单击“Add”，完成添加模型文件。

   添加模型文件后，可按如下目录在“Project”窗口查看模型文件。
   ```
   ├── 工程名
   │   ├── model
   │   │   ├──*.om          //模型文件
   ```

步骤4 准备推理数据。

请参照创建的应用工程根目下的readme.md文件中的“编译运行”下的“准备输入图片”，准备输入图片。

----结束

7.3.3 添加自定义动态链接库（可选）

用户应用中经常需要使用其他第三方动态链接库（例如使用OpenCV对YOLOv3的推理结果进行后处理）。在开发环境与运行环境部署在不同的机器的场景下，若用户仅能通过MindStudio使用运行环境，则远程运行会由于无法找到对应动态链接库而失败。为解决这一问题，在MindStudio的应用开发中增加对用户自定义动态链接库的支持，提高客户开发效率。

用户添加自定义动态链接库的步骤参考如下：

步骤1 完成动态链接库的交叉编译。

具体交叉编译步骤请参见使用的动态链接库的对应文档，需要注意交叉编译器应与运行环境系统架构相对应。

步骤2 新建存放动态链接库的目录，将交叉编译出的动态链接库拷贝或软链接至该目录下，目录名称可由用户自定义（如图7-13中lib目录下的libcustom.so）。

图 7-13 存放动态链接库的目录

需要注意：新建的存放动态链接库的目录，需要在步骤2中配置应用工程运行参数时进行环境变量的配置。环境变量的配置是以可执行文件main.cpp所在位置“*工程名*/out/main.cpp”为基准，即截图中配置的../mylib意为，mylib为存放动态链接库的目录名称，文件夹位置在工程名目录下。有以下两种方式配置：

- 在Environment Variables处直接输入，如图图7-14。

图 7-14 动态链接库目录

- 单击后在出的弹窗内单击+，填写。

图 7-15 环境变量填写

----结束

7.3.4 开发应用

创建 Empty Project 工程时

请参见表7-1中《应用软件开发指南（C&C++）》的说明开发应用。

创建 Sample ResNet-50 工程时

- 创建应用工程后，工程的src目录下自带应用的模板代码，包含系统初始化、模型执行、模型卸载、资源销毁等代码，若想直接使用src目录下的*.cpp模板代码，则只需按照模型文件和图片的名称、所在的路径修改sample_process.cpp中的如下代码，此处只能设置相对路径，不能设置绝对路径。
  - 请根据实际情况，修改图片文件的名称以及路径。
    ```
    //此处的..表示相对路径，相对可执行文件所在的目录
    //例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
    string testFile[] = {
            "../data/dog1_1024_683.bin",
            "../data/dog2_1024_683.bin"
      };
    ```
  - 请根据实际情况，修改*.om模型文件的名称。
    ```
    //此处的..表示相对路径，相对可执行文件所在的目录
    //例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
    const char* omModelPath = "../model/resnet50.om";
    ```

  在模板代码中，“接口名称以acl开头”的接口是系统对用户开放的接口，关于接口的详细说明，可参见表7-1中《应用软件开发指南（C&C++）》中的“AscendCL API参考”。
- 创建应用工程后，若不想使用工程中的模板代码，您可以查看表7-1中《应用软件开发指南（C&C++）》中的说明开发应用。

7.3.5 编译应用工程

步骤1 （可选）修改工程目录下的“src/CMakeLists.txt”文件。

- include_directories：添加头文件所在的目录。

  示例如下：

```
include_directories(
directoryPath1
directoryPath2
)
```

- link_directories：添加库文件所在的目录。

  示例如下：

```
link_directories(
directoryPath3
directoryPath4
)
```

- add_executable：添加*.cpp文件所在的目录。

  示例如下：

```
add_executable(
main
directoryPath5
directoryPath6
)
```

- target_link_libraries：添加编译应用依赖的库文件。

  示例如下：

```
target_link_libraries(
main
ascendcl
libName1
libName2)
```

- install：选择编译产物main安装到指定路径。

  示例如下：

```
install(TARGETS main DESTINATION ${CMAKE_RUNTIME_OUTPUT_DIRECTORY})
```

说明

关于cmake参数的详细介绍，请参见**https://cmake.org/cmake/help/latest/guide/tutorial/index.html**，选择对应版本后查看参数。

步骤2 在MindStudio工程界面，依次选择“Build > Edit Build Configuration...”，进入编译配置页面。

1. 配置编译参数。

   如果编译后的可执行文件在运行环境上运行，Target OS下拉框处显示的选项与所安装的acllib组件有关，您可以在ADK界面查看acllib组件的基本信息，如何进入ADK界面，请参见**ADK管理-概述**。

   在ADK界面的ACLlib组件所在行，根据Host OS Arch列显示的操作系统及其架构，在Target OS下拉框处选择操作系统，在Target Architecture下拉框处选择架构。

图 7-16 编译配置

2. 单击“Build”编译工程。

   如果在编译过程中无错误提示，且提示“build successfully”，则表示编译成功，您可以在如下目录中查看可执行文件main。

```
├── 工程名
│   ├── out
│   │   ├──main        //可执行文件
```

----结束

7.3.6 运行应用工程

步骤1 在MindStudio工程界面，依次选择“Run > Edit Configurations...”，进入运行配置页面。

步骤2 选择“Ascend APP > *工程名*”配置应用工程运行参数，参数说明如表7-5所示。

表 7-5 应用工程运行配置参数说明

配置参数	参数说明
Name	运行配置名称，可修改。
Executable	选择可执行文件的位置。
Run Mode	● Remote Run：远程运行。 ● Local Run：本地运行。
SSH Connection	SSH连接，选择Remote Run模式时可见，配置与修改具体请参见13.1 SSH连接管理。 下拉选择远程运行应用的服务器的IP地址，若未添加，请单击+进行添加，详情请参见13.1.2 新增SSH连接。
Command Arguments	运行应用传入的参数，和用户的代码实现有关，为可选配置。
Environment Variables	环境变量配置。可以直接在框中输入也可以单击 后在出的弹窗内单击 +，填写。

配置参数	参数说明
Toolkit Path	Toolkit软件包安装路径。选择Remote Run模式时可见，一般为可选配置，Profiling性能分析时为必选项。
Enable Profiling	是否开启Profiling相关功能，详细配置，请参见**10.1 应用工程Profiling**。
Before launch	此栏中可配置运行前的预处理操作，为可选配置。 例如，单击 + 添加“Ascend APP Build”，在运行可执行文件前，编译可执行文件。

步骤3 配置完成后，单击“Apply”保存运行配置，单击“OK”，关闭运行配置窗口。

步骤4 在MindStudio工程界面，依次选择“Run > Run...”。

在弹出框中选择已创建好的运行配置信息，运行应用。

- 如果在运行过程中无错误提示，且提示“Running *** finished”，则表示运行结束。

  此处以运行acl_resnet50样例为例，运行结束后，直接查看运行结果（top5置信度的类别），不生成推理结果文件。

  运行结果示例如**图7-17**，index及其value会根据运行环境的实际情况有所不同：

图 7-17 运行结果

```
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  get run mode success
[INFO]  load model ../model/resnet50_0528.om success
[INFO]  create model description success
[INFO]  create model output success
[INFO]  start to process file:../data/dog1_1024_683.bin
[INFO]  model execute success
[INFO]  top 1: index[161] value[0.767578]
[INFO]  top 2: index[162] value[0.154785]
[INFO]  top 3: index[167] value[0.038513]
[INFO]  top 4: index[163] value[0.021606]
[INFO]  top 5: index[166] value[0.011658]
[INFO]  output data success
[INFO]  start to process file:../data/dog2_1024_683.bin
[INFO]  model execute success
[INFO]  top 1: index[267] value[0.936035]
[INFO]  top 2: index[266] value[0.041138]
[INFO]  top 3: index[265] value[0.018845]
[INFO]  top 4: index[219] value[0.002609]
[INFO]  top 5: index[160] value[0.000295]
[INFO]  output data success
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device is 0
[INFO]  end to finalize acl
```

- 如果运行过程中有错误提示，且需要查看运行环境的详细日志时，请参见**表7-1**中的《日志参考》：
  - 可参见“日志文件介绍”查看日志，日志文件路径默认为“$HOME/ascend/log”。
  - 可参见“设置日志级别”查看或修改日志级别（默认为ERROR）。

----结束

7.4 应用工程调试

7.4.1 调试前准备

安装调试依赖项

- 为支持多交叉架构的调试场景，需要在安装MindStudio的服务器（UI Host）上安装gdb-multiarch，以下为在Ubuntu环境上安装的命令示例：

  apt install gdb-multiarch

说明

执行命令**gdb-multiarch**，若返回gdb-multiarch版本信息则表明安装成功。若出现缺失so文件的错误，需要手动安装缺失的so文件。

- 如果选择在远程调试，需要在连接的远端服务器（AI Host）上安装gdbserver并在安装MindStudio的服务器（UI Host）上安装ssh-askpass。以下为在Ubuntu环境上安装的命令示例：

  apt-get install gdbserver

  apt install ssh-askpass

  若gdbserver不能直接安装，则需要编译gdbserver。操作步骤如下：

  a. 获取gdb源码（**获取链接**），建议获取gdb-8.1.1.tar.gz。

  b. 请用户按照实际需要获取并安装交叉编译器。

  c. 交叉编译gdbserver，操作步骤如下，以gdb-8.1.1.tar.gz为例。

     i. 解压gdb源码压缩包，并进入gdbserver的子目录中：

     tar xf gdb-8.1.1.tar.gz

     cd gdb-8.1.1/gdb/gdbserver

     ii. 配置编译参数：

     ./configure --host=*{compiler}* **--prefix=***$HOME/install/gdbserver*

     *{compiler}*为用户获取的交叉编译器，请用户自行替换。*$HOME/install/gdbserver*为指定安装目录示例。

     iii. 编译及安装：

     make -j

     make install

  d. 部署gdbserver。

     将编译好的gdbserver拷贝至目标运行服务器。

说明

上述gdb-multiarch和gdbserver软件仅用于调试用户开发的程序，不存在安全风险。

配置编译与运行环境

步骤1 在工具栏选择**Build>Edit Build Configuration...**进入编译配置界面。在Build Type处选择Debug，单击Build进行编译，生成用于debug的可执行文件同时工具栏中debug图标 将点亮。编译生成的可执行文件在“*工程名*/out”目录下。

步骤2 在工具栏选择**Run>Edit Configurations...**进入运行配置界面。

- Executable：选择编译用可执行文件的路径，选择到文件一级。
- Run Mode：选择本地调试（Local Run）或者远程调试（Remote Run）。

步骤3 如果选择本地调试，单击Apply后单击OK结束配置。

如果选择远程调试，需要在SSH Connection处配置远程连接的服务器，单击Apply后单击OK结束配置（具体请参见**13.1 SSH连接管理**）。

----结束

7.4.2 断点管理

断点是指在某一行设置一个断点，当程序运行到断点处时暂停执行，然后用户会查看变量、内存、检查程序逻辑是否正确。您可以设置各种方式的断点，例如让程序暂停在函数执行的入口处，或者让程序暂停在文件的某一行等。

添加断点

打开想要调试的代码，选定要设置断点的代码行，在行号的区域后面单击鼠标左键，既可以添加断点，如图7-18所示，断点添加成功后，左侧代码行号处会显示●图标。

图 7-18 添加断点

删除断点

在设置断点的地方，单击左侧的●即可取消断点。

Disable 断点

若用户设置了多个断点，执行调试时想使用部分断点又不删除其他断点，则可以通过该功能将暂不使用的断点Disable，操作方法如下：

右击已经设置断点的代码行号处的●，在弹出界面中去勾选Enabled，可以看到原有●断点标识变成○，则再执行调试功能时，该断点不生效。

图 7-19 断点 Disable

Enable 断点

若已经有部分断点设置了Disable，执行新的调试功能时想使原有Disable的断点生效，则可以通过该功能实现。操作方法如下：

右击已经Disable断点处的○，在弹出界面中勾选Enabled，可以看到原有○断点标识变成●，则后续执行调试功能时，该断点生效。

图 7-20 断点 Enable

MindStudio重启以及工程重启后，不会影响当前工程中已经设置的断点状态和数量。

查看断点

通过该功能，用户可以查看曾经设置过的断点。

单击Disable断点或Enable断点界面的“More(Ctrl+Shift+F8)”，弹出图7-21所示断点查看界面。

图 7-21 断点查看界面

左侧序号1区域展示了当前代码中所设置的所有断点，序号2对应相应断点的Disable或Enable功能，序号3为代码展示区域。

- 单击序号1对应断点左侧的复选框，勾选表示该断点Enable，相应序号2处Enable功能使能，同时序号3对应代码行左侧行号处显示 ● 图标；去勾选表示该断点Disable，相应序号2处Enable功能去使能，同时序号3对应代码行左侧行号处显示 ○ 图标。
- 选中序号1处的断点，单击上方的 − 可以删除断点，同时序号3代码展示区域，相应代码行号处的断点图标随之删除。
- 在序号3代码展示区域，相应代码行号处单击，则该行被标记为断点，同时序号1处会自动增加该断点信息。

将变量添加到 Variables 区域

选中要查看的变量，右击选择“Add to Watches”，可以看到变量添加到下方“Variables”区域中，如图7-22所示。

图 7-22 将变量添加到 Variables 区域

您也可以单击左侧工具栏的功能按钮，完成更多功能：

- 单击左侧工具栏+添加新的变量到“Variables”区域中。
- 单击左侧工具栏−删除选中的变量。
- 单击左侧工具栏复制选中变量，用户可以基于复制后的变量，修改成想要的参数：
  选中复制后的变量，右击选择“Edit”，重新输入新的变量名即可。
- 若Variables区域变量或线程太多不方便查看变量信息时，可以单击左侧工具栏打开新的“Watches”窗口，变量会自动呈现到该窗口中，如图7-23所示。单击Watches窗口上方的，Watches窗口会自动合入Variables窗口。

  图 7-23 开启 Watches 窗口。

7.4.3 调试执行

启动调试

1. 设置完断点后，单击工具栏中debug图标即可启动调试功能。

   说明

   - 启动调试功能时，MindStudio需要进行模型文件、数据文件以及调试依赖等文件的加载，所以启动过程会比较慢，大约需要1分钟左右。
   - 远程调试的情况下，单击工具栏中debug图标需要再次输入远程连接的服务器的登陆密码。

2. MindStudio界面停下来时，会停在设置的第一个断点处，调试界面如图7-24所示。工程右下方出现Debugger视图，其中：
   - 左侧Frames区域是程序的方法调用栈，在该区域中显示了程序执行到断点处所调用过的所用方法，越下面的方法被调用的越早。红框中的信息表示当前调试程序停留的代码行，例如*main.cpp*程序的第17行。
   - 右侧Variables展示变量信息以及线程信息。

图 7-24 调试工程界面

调试窗口图标解释如表7-6所示。

表 7-6 调试窗口图标说明

图标	名字	作用
	Rerun(Ctrl +F5)	重新执行调试功能。
	Resume program(F9)	继续运行程序，到下一个断点处。 运行一个断点到下一个断点之间需要执行的代码，如果后面代码没有断点，再次单击该按钮将会执行完程序。
	Stop(Ctrl+F2)	终止调试功能。
	View Breakpoints(Ctrl+Shift+F8)	查看所有断点。 查看曾经设置过的断点并设置断点的一些属性，弹出界面以及功能详细说明请参见**查看断点**。
	Mute Breakpoints	暂停使用所有断点。 请参见**Mute Breakpoints功能**。

图标	名字	作用
	Step Over(F8)	进入下一行代码。 单击该按钮，程序向下执行一行，如果当前行有函数调用，该函数将被执行完毕返回，然后到下一行，同时会显示frame信息和变量信息，详细操作请参见**Step Over功能**。
	Step Into(F7)	进入函数。 程序向下执行一行。如果该行有自定义函数，则运行进入自定义函数（不会进入官方库的函数），同时会显示frame信息和变量信息，详细操作请参见**Step Into功能**。
	Force Step Into(Alt+Shift+F7)	强制进入函数。 单击按钮，调试时能进入任何函数，包括自定义函数以及官方库函数，
	Step Out(Shift+F8)	跳出函数。 如果调试时进入了某个函数，若检查该函数没有问题，则可以使用Step Out按钮跳出该函数，返回到原始位置。详细操作请参见**Step Out功能**。
	Run to cursor(Alt+F9)	运行到光标出。 可以在没有设置断点的情况下，让程序运行到光标所在行时暂停，同时会显示frame信息和变量信息，详细操作请参见**Run to cursor功能**。

说明

- 当前仅支持对C++文件进行调试。
- 同时运行多个C++文件，启动调试、停止调试、单步调试、进入函数、跳出函数等功能都仅对当前debug界面选择的文件起作用。

下面以具体示例介绍各个图标功能。

Step Over 功能

1. 设置断点。
2. 启动调试功能，界面停在断点处。
3. 单击 程序向下执行一行。详细示例如**图7-25**所示。

图 7-25 Step Over 功能

Step Into 功能

1. 设置断点。
2. 启动调试功能，界面停在断点处。
3. 单击程序向下执行一行。如果该行有自定义方法，则运行进入自定义方法，详细示例如图7-26所示。

图 7-26 Step Into 功能

Step Out 功能

若执行调试时，通过**Step Into功能**进入了一个函数，例如“GetFileList”，若检查该函数没有问题，则可以通过单击跳出该函数，返回到原来位置，如**图7-27**所示。

图 7-27 Step Out 功能

Run to cursor 功能

若某行代码没有设置断点，用户想调试该行代码，则可以通过单击 ，将断点移到光标所在代码行，如图7-28所示。

图 7-28 Run to cursor 功能

Mute Breakpoints 功能

调试过程中，如果想暂停使用所有断点，单击按钮后，所有断点被置灰，如图7-29所示，同时该按钮显示灰色底纹，单击调试代码直接运行完成。

图 7-29 Mute Breakpoints 功能

```
Result SampleProcess::Process()
{
    // model init
    ModelProcess modelProcess;
    const char* omModelPath = "../model/resnet50.om";
    Result ret = modelProcess.LoadModel(omModelPath);
    if (ret != SUCCESS) {
        ERROR_LOG("execute LoadModel failed");
        return FAILED;
    }

    ret = modelProcess.CreateModelDesc();
    if (ret != SUCCESS) {
        ERROR_LOG("execute CreateModelDesc failed");
        return FAILED;
    }
```

8 自定义算子开发

说明

8.1 使用前必读
8.2 开发流程
8.3 算子分析
8.4 工程创建
8.5 TBE算子开发（TensorFlow）
8.6 TBE算子开发（PyTorch）
8.7 TBE算子开发（MindSpore）
8.8 AI CPU算子开发（TensorFlow）
8.9 其他功能及操作

8.1 使用前必读

本章介绍如何使用MindStudio工具实现不同框架的算子开发。包括MindSpore、Pytorch、TensorFlow的TBE算子开发和TensorFlow的AI CPU算子开发。此过程中，我们主要介绍**MindStudio工具操作部分**，对**代码实现部分**仅做基本介绍，如需了解更多代码开发技巧，请参考对应的算子开发指南。

表 8-1 参考手册

产品	手册
Atlas 200 DK	《**TBE自定义算子开发指南**》

版权所有 © 华为技术有限公司

产品	手册
其他产品	• 《TBE自定义算子开发指南(推理)》 • 《TBE自定义算子开发指南(训练)》 • 《AI CPU自定义算子开发指南(推理)》 • 《AI CPU自定义算子开发指南(训练)》 • MindSpore官方教程>自定义算子（Ascend）

MindStudio工具提供了算子开发样例，获取路径为MindStudio安装目录下的“/MindStudio/samples/operator_demo_projects”。算子开发样例包括：

- MindSpore算子工程的Square TBE算子，工程目录为“/mindspore_operator_sample”。
- TensorFlow算子工程的Add TBE算子和ReshapeCust AI CPU算子，工程目录分别为“/tbe_operator_sample”和“/aicpu_operator_sample”。

Ascend/ascend-toolkit/*xxx*/opp/op_impl/built-in/ai_core/tbe/impl目录下为昇腾AI处理器支持的内置AI Core算子的实现文件，开发者在进行算子代码实现时可进行参考。

8.2 开发流程

8.2.1 TensorFlow TBE 算子开发流程

通过MindStudio工具进行TensorFlow TBE算子开发的总体开发流程如图8-1所示。

图 8-1 TensorFlow TBE 算子开发流程图

1. 算子分析：确定算子功能、输入、输出，算子开发方式、算子OpType以及算子实现函数名称等。
2. 工程创建。

   通过MindStudio工具创建TBE算子工程，创建完成后，会自动生成算子工程目录及相应的文件模板，开发者可以基于这些模板进行算子开发。
3. 算子开发。
   - 算子代码实现：描述算子的实现过程。
   - 算子原型定义：算子原型定义规定了在昇腾AI处理器上可运行算子的约束，主要包含定义算子输入、输出、属性和取值范围，基本参数的校验和shape的推导，原型定义的信息会被注册到GE的算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。
   - 算子信息定义：算子信息配置文件用于将算子的相关信息注册到算子信息库中，包括算子的输入输出dtype、format以及输入shape信息。网络运行时，FE会根据算子信息库中的算子信息做基本校验，判断是否需要为算子插入合适的转换节点，并根据算子信息库中信息找到对应的算子实现文件进行编译，生成算子二进制文件进行执行。
   - 算子适配插件实现：基于第三方框架（TensorFlow/Caffe）进行自定义算子开发的场景，开发人员完成自定义算子的实现代码后，需要进行插件的开发将基于Tensorflow/Caffe的算子映射成适配昇腾AI处理器的算子，将算子信息注册到GE中。基于TensorFlow/Caffe框架的网络运行时，首先会加载并调用GE中的插件信息，将TensorFlow/Caffe网络中的算子进行解析并映射成昇腾AI处理器中的算子。
   - 算子实现验证。

     通过测试用例来验证算子实现代码的功能正确性和逻辑正确性。

     - UT测试：即单元测试（Unit Test），仿真环境下验证算子实现的功能正确性，包括算子逻辑实现代码及算子原型定义实现代码。
     - ST测试：即系统测试（System Test），可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性。
4. 算子编译。

   将算子插件实现文件、算子原型定义文件、算子信息定义文件编译成算子插件、算子原型库、算子信息库。
5. 算子部署。

   将算子实现文件、编译后的算子插件、算子原型库、算子信息库部署到昇腾AI处理器算子库，为后续算子在网络中运行构造必要条件。

8.2.2 PyTorch TBE 算子开发流程

通过MindStudio工具进行PyTorch TBE算子开发的总体开发流程如**图8-2**所示。

图 8-2 PyTorch TBE 算子开发流程图

1. 算子分析：确定算子功能、输入、输出，算子开发方式、算子OpType以及算子实现函数名称等。
2. 工程创建。

   通过MindStudio工具创建TBE算子工程，创建完成后，会自动生成算子工程目录及相应的文件模板，开发者可以基于这些模板进行算子开发。
3. 算子开发。
   - 算子实现：描述算子的实现过程。
   - 算子原型定义：算子原型定义规定了在昇腾AI处理器上可运行算子的约束，主要包含定义算子输入、输出、属性和取值范围，基本参数的校验和shape的推导，原型定义的信息会被注册到GE的算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。
   - 算子信息定义：算子信息配置文件用于将算子的相关信息注册到算子信息库中，包括算子的输入输出dtype、format以及输入shape信息。网络运行时，FE会根据算子信息库中的算子信息做基本校验，判断是否需要为算子插入合适的转换节点，并根据算子信息库中信息找到对应的算子实现文件进行编译，生成算子二进制文件进行执行。
   - 算子实现验证。

     通过测试用例来验证算子实现代码的功能正确性和逻辑正确性。

     - UT测试：即单元测试（Unit Test），仿真环境下验证算子实现的功能正确性，包括算子逻辑实现代码及算子原型定义实现代码。
     - ST测试：即系统测试（System Test），可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性。
4. 算子编译。

将算子插件实现文件、算子原型定义文件、算子信息定义文件编译成算子插件、算子原型库、算子信息库。

5. 算子部署。

   将算子实现文件、编译后的算子插件、算子原型库、算子信息库部署到昇腾AI处理器算子库，为后续算子在网络中运行构造必要条件。

6. PyTorch算子适配。

   NPU具有内存管理，设备管理，以及算子调用实现。PyTorch算子适配根据PyTorch原生结构进行NPU扩展。

8.2.3 MindSpore TBE 算子开发流程

通过MindStudio工具进行TBE算子开发的总体开发流程如图8-3所示

图 8-3 MindSpore TBE 算子开发流程

1. 算子分析：确定算子功能、输入、输出，算子开发方式、算子OpType以及算子实现函数名称等。
2. 工程创建。

   通过MindStudio工具创建MindSpore TBE算子工程，创建完成后，会自动生成算子工程目录及相应的文件模板，开发者可以基于这些模板进行算子开发。

3. 算子开发。
   - 算子和注册信息文件：通过TBE（Tensor Boost Engine）提供的特性语言接口，描述算子内部计算逻辑的实现。注册信息描述TBE算子的基本信息，如算子名称、支持的输入输出类型等。算子信息是指导后端选择算子实现的关键信息，同时也指导后端为算子插入合适的类型和格式转换。
   - 算子原型定义：定义了算子在网络中的前端接口原型，也是组成网络模型的基础单元，主要包括算子的名称、属性（可选）、输入输出名称、输出shape推理方法、输出dtype推理方法等信息。
   - 算子实现验证。

     通过测试用例来验证算子实现代码的功能正确性和逻辑正确性。

     - UT测试：即单元测试（Unit Test），仿真环境下验证算子实现的功能正确性，包括算子逻辑实现代码及算子原型定义实现代码。

- ST测试：即系统测试（System Test），可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性。

8.2.4 TensorFlow AI CPU 算子开发流程

通过MindStudio工具进行TensorFlow AICPU算子开发的总体开发流程如图8-4所示。

图 8-4 TensorFlow AICPU 算子开发流程

1. 算子分析：明确算子的功能、输入、输出，规划算子类型名称以及算子编译生成的库文件名称等。
2. 工程创建。

   通过MindStudio工具创建TensorFlow AICPU算子工程，创建完成后，会自动生成算子工程目录及相应的文件模板，开发者可以基于这些模板进行算子开发。
3. 算子开发。
   - 算子实现：实现算子的计算逻辑。
   - 算子原型定义：算子原型定义规定了在昇腾AI处理器上可运行算子的约束，主要体现算子的数学含义，包含定义算子输入、输出、属性和取值范围，基本参数的校验和shape的推导，原型定义的信息会被注册到GE的算子原型库中。离线模型转换时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。
   - 算子信息定义：算子信息配置文件用于将算子的相关信息注册到算子信息库中，包括算子的OpType、输入输出dtype、name等信息。网络运行时，AI CPU Engine会根据算子信息库中的算子信息做基本校验，并进行算子匹配。
   - 算子适配插件实现：基于第三方框架（TensorFlow/Caffe）进行自定义算子开发的场景，开发人员完成自定义算子的实现代码后，需要进行插件的开发将基于TensorFlow/Caffe的算子映射成昇腾AI处理器的算子。
4. 算子编译。

   将算子适配插件实现文件、原型定义文件、信息定义文件编译成算子插件库文件、算子原型库、算子信息库。

5. 算子部署。

   将算子实现文件、插件库文件、原型库、信息库部署到算子库中（opp的对应目录下）。

6. 算子实现验证。

   ST测试：即系统测试（System Test），可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性。

8.3 算子分析

简介

本节以Add TBE算子和AICPU算子开发为例进行算子分析，帮助用户快速掌握算子分析流程。MindSpore算子工程的Square TBE算子分析请用户自行完成。

TBE 算子分析

使用TBE DSL方式开发Add算子前，我们需要确定算子功能、输入、输出，算子开发方式、算子类型以及算子实现函数名称等。

步骤1 明确算子的功能以及数学表达式。

add算子的数学表达式为：

```
z=x+y
```

计算过程是：将两个输入参数相加，得到最终结果z并将其返回。

步骤2 明确输入和输出。

- add算子有两个输入：x与y，输出为z。
- 本样例中算子的输入支持的数据类型为float16、float32、 int32，算子输出的数据类型与输入数据类型相同。
- 算子输入支持所有shape，输出shape与输入shape相同。
- 算子输入支持的format为：NCHW,NC1HWC0,NHWC,ND。

步骤3 确定算子开发方式及使用的计算接口。

1. 计算过程只涉及加法操作，初步分析可使用te.lang.cce.vadd(lhs, rhs)接口实现x+y，请查看
   - 《**TBE自定义算子开发指南(推理)**》
   - 《**TBE自定义算子开发指南(训练)**》
2. 由于te.lang.cce.vadd(lhs, rhs)接口要求两个输入tensor的shape需要相同，所以需要首先获取两个输入tensor中较大的shape值，然后调用te.lang.cce.broadcast(var, shape, output_dtype=None)接口将入参广播成指定的shape大小。

步骤4 明确算子实现文件名称、算子实现函数名称以及算子的类型（OpType）。

算子命名规则如下：

- 算子类型需要采用大驼峰的命名方式，即采用大写字符区分不同的语义。
- 算子文件名称和算子函数名称，可选用以下任意一种命名规则：

- 用户自定义，此时需要在**算子信息定义**中配置**opFile.value**与**opInterface.value**。
- 不配置**算子信息定义**中**opFile.value**与**opInterface.value**，FE会将OpType按照如下方式进行转换后进行算子文件名和算子函数名的匹配。
  - 首字符的大写字符转换为小写字符。
    例如：Abc -> abc
  - 小写字符后的大写字符转换为下划线+小写字符。
    例如：AbcDef -> abc_def
  - 紧跟数字以及大写字符后的大写字符，作为同一语义字符串，查找此字符串后的第一个小写字符，并将此小写字符的前一个大写字符转换为下划线+小写字符，其余大写字符转换为小写字符。若此字符串后不存在小写字符，则直接将此字符串中的大写字符转换为小写字符。
    例如：ABCDef -> abc_def；Abc2DEf -> abc2d_ef；Abc2DEF -> abc2def；ABC2dEF -> abc2d_ef。

因此本例中，算子类型定义为Add；算子的实现文件名称及实现函数名称将首字母转换小写字符，定义为add。

通过以上分析，得到Add算子的设计规格如下：

表 8-2 Add 算子设计规格

算子类型（OpType）	Add			
算子输入	name：x1	shape：all	data type：float16、float32、int32	format：NCHW,NC1HWC0,NHWC,ND
	name：x2	shape：all	data type：float16、float32、int32	format：NCHW,NC1HWC0,NHWC,ND
算子输出	name：y	shape：all	data type：float16、float32、int32	format：NCHW,NC1HWC0,NHWC,ND
算子实现使用的主要DSL接口	te.lang.cce.broadcast(var, shape, output_dtype=None) te.lang.cce.vadd(lhs, rhs)			
算子实现文件/实现函数名称	add			

----结束

AI CPU 算子分析

使用AI CPU方式开发算子前，我们需要确定算子功能、输入、输出、算子类型以及算子实现函数名称等。

步骤1 明确算子的功能以及数学表达式。

以Add算子为例，Add算子的数学表达式为：

```
z=x+y
```

计算过程是：将两个输入参数相加，得到最终结果z并将其返回。

步骤2 明确输入和输出。

- 例如Add算子有两个输入：x与y，输出为z。
- 本样例中算子的输入支持的数据类型为float16、float32、 int32，算子输出的数据类型与输入数据类型相同。
- 算子输入支持所有shape，输出shape与输入shape相同。
- 算子输入支持的format为：NCHW,NHWC,ND。

步骤3 明确算子实现文件名称以及算子的类型（OpType）。

- 算子类型需要采用大驼峰的命名方式，即采用大写字符区分不同的语义。
- 算子文件名称，可选用以下任意一种命名规则：

  建议将OpType按照如下方式进行转换，得到算子文件名称。

  转换规则如下：

  – 首字符的大写字符转换为小写字符。

    例如：Abc -> abc

  – 小写字符后的大写字符转换为下划线+小写字符。

    例如：AbcDef -> abc_def

  – 紧跟数字以及大写字符后的大写字符，作为同一语义字符串，查找此字符串后的第一个小写字符，并将此小写字符的前一个大写字符转换为下划线+小写字符，其余大写字符转换为小写字符。若此字符串后不存在小写字符，则直接将此字符串中的大写字符转换为小写字符。

    例如：ABCDef -> abc_def；Abc2DEf -> abc2d_ef；Abc2DEF -> abc2def；ABC2dEF -> abc2d_ef。

因此本例中，算子类型定义为Add，算子的实现文件名称为add，因此各个交付件的名称建议命名如下：

- 算子的代码实现（即kernel实现）文件命名为add_kernel.h与add_kernel.cc。
- 插件实现文件命名为add_kernel_plugin.cc。
- 原型定义文件命名为：add.h与add.cc。
- 信息定义文件命名为add.ini。

通过以上分析，得到Add算子的设计规格如下：

表 8-3 Add 算子设计规格

算子类型（OpType）	Add		
算子输入	name：x	shape：all	data type： float16、float32、 int32

	name：y	shape：all	data type： float16、float32、 int32
算子输出	name：z	shape：all	data type： float16、float32、 int32
算子实现文件名称	add		

----结束

8.4 工程创建

8.4.1 创建算子工程

简介

本节介绍如何通过MindStudio工具创建算子工程，创建完之后工具会自动生成算子工程目录及相应的文件模板，您可以基于这些模板进行算子开发。

操作步骤

步骤1 进入算子工程创建界面。

- 首次登录MindStudio：在MindStudio欢迎界面中单击“Create new project”，进入创建工程界面。
- 非首次登录MindStudio：在顶部菜单栏中选择“File > New > Project...”，进入创建工程界面。

步骤2 创建算子工程。

1. 在左侧导航栏选择“Ascend Operator”，在右侧配置算子工程信息，配置示例如表8-4所示。

表 8-4 工程信息配置

参数	参数说明	示例
Name	工程名称，用户自行配置。 名称必须以字母开头，只能包含字母、数字、中划线和下划线，且长度不能超过64个字符。	MyOperator
Description	工程描述信息，自行配置。	可选配置
ADK Version	当前ADK的版本号。	选择当前ADK的版本号
Project Location	工程的存储路径。	保持默认

2. 单击“Next”，在弹出的页面中配置算子相关信息，如表8-5所示。

表 8-5 算子信息配置

参数	参数说明	示例
New Operator（算子创建方式）		
Empty	表示创建空的算子工程。 选择此选项，下方会显示“Operator Type”配置项，请在此处输入需要创建的算子的类型，请根据8.3 算子分析进行配置。	三选一
From Template	表示基于IR定义模板创建算子工程。 选择此选项，下方会显示“Template File”配置项，用户需要选择定义了IR原型的Excel文件。 **须知** 请从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-*linux*/toolkit/tools/msopgen/template**目录下获取IR定义模板Ascend_IR_Template.xlsx，请在模板文件中的“Op”页签修改自定义算子的IR定义，IR定义的配置请参见**IR定义配置说明**。	
From Tensorflow	表示基于Tensorflow原型定义创建算子工程。 选择此方式创建算子工程，请先在安装MindStudio的服务器中下载**Tensorflow源码**。选择此选项后，下方会出现详细配置栏。 – Operator Path：请选择Tensorflow源码所在目录，为提升搜索效率，建议选择到tensorflow/core/ops目录。 – Operator Type：请配置为需要创建算子的Op Type。	
Plugin Framework	算子所在模型文件的框架类型。 – MindSpore – PyTorch – TensorFlow	TensorFlow

参数	参数说明	示例
Compute Unit	– “Plugin Framework”选择“MindSpore”，仅支持选择“AI Core/Vector Core” – 若“Plugin Framework”选择“PyTorch”，仅支持选择“AI Core/Vector Core”。 – 若“Plugin Framework”选择“TensorFlow”，有以下两种选项： ▪ AI Core/Vector Core：算子运行在AI Core或者Vector Core上，代表TBE算子。 ▪ AI CPU：算子运行在AI CPU上，代表AI CPU算子。	AI Core/Vector Core
Unit Type	– 当“Compute Unit”选择“AI Core/Vector Core”时显示此配置项。 – “Plugin Framework”选择“MindSpore”时不显示此配置项。	请根据实际昇腾AI处理器版本下拉选择算子计算单元。

3. 单击“Finish”，完成算子工程的创建。

   若工作窗口已打开其他工程，会出现如图8-5所示提示。

   – 选择“This Window”，则直接在当前工作窗口打开新创建的工程。
   – 选择“New Window”，则新建一个工作窗口打开新创建的工程。

图 8-5 工程创建提示

须知

– 新建工程的默认存储路径为安装用户家目录的“AscendProjects”文件夹下。
– 若开发者需要自定义多个算子，需要在同一算子工程中进行实现。可选中算子工程根目录，右键选择**New > Operator** 增加算子。
– 同一算子工程中，暂时不支持AICPU算子和TBE算子同名。

步骤3 查看算子工程目录结构和主要文件。

框架为TensorFlow、PyTorch时的工程目录结构和主要文件。

```
├── .idea
├── build                          //编译生成的中间文件
├── cmake                          //编译相关公共文件存放目录、
├── cpukernel                      //AI CPU算子文件目录，TensorFlow TBE算子和PyTorch算子工程不会创建此目录
│   ├── impl                       //算子实现文件目录
│   │   ├── reshape_cust_kernel.cc  //算子源文件
```

```
│   │   ├── reshape_cust_kernel.h        //算子头文件
│   ├── op_info_cfg                      //算子信息库文件目录
│   │   ├── aicpu_kernel
│   │   │   ├── reshape_cust.ini         //算子信息定义文件
│   ├── CMakeLists.txt                   //编译规则文件，会被算子工程根目录中的CMakeLists.txt文件调用
│   ├── toolchain.cmake
├── framework                            //算子插件实现文件目录
│   ├── tf_plugin                        //存放tensorflow框架的算子插件文件及编译规则文件，若“Plugin Framework”为“PyTorch”，不会创建此目录
│   │   ├── tensorflow_add_plugin.cpp    //add为算子类型，TBE算子示例为add，AI CPU算子为reshape_cust
│   │   ├── CMakeLists.txt               //编译规则文件，会被上级目录中的CMakeLists.txt文件调用
│   ├── CMakeLists.txt                   //编译规则文件，会被算子工程根目录中的CMakeLists.txt文件调用
├── op_proto                             //算子IR定义文件目录
│   ├── add.cpp                          //add为算子类型，TBE算子示例为add，AI CPU算子为reshape_cust
│   ├── add.h                            //add为算子类型，TBE算子示例为add，AI CPU算子为reshape_cust
│   ├── CMakeLists.txt                   //编译规则文件，会被算子工程根目录的CMakeLists.txt文件调用
├── scripts                              //工程相关脚本
├── tbe                                  //TBE算子文件目录，TensorFolw AI CPU算子工程不会创建此目录
│   ├── impl                             //算子实现文件目录
│   │   ├── add.py
│   ├── op_info_cfg                      //算子信息库文件目录
│   │   ├── ai_core
│   │   │   ├── {soc_version}
│   │   │   │   ├── add.ini
├── .project                             //工程信息文件，包含工程类型、工程描述、运行目标设备类型以及ADK版本
├── CMakeLists.txt
├── MyOperator.iml
```

框架为MindSpore时的工程目录结构和主要文件。

```
├── .idea
├── mindspore
│   ├── impl                 //算子实现文件目录
│   │   ├── cus_square_impl.py      //实现算子和注册算子信息 py文件
├── op_proto
│   ├── cus_square.py                //算子原语py文件
├── .project                 //工程信息文件，包含工程类型、工程描述、运行目标设备类型以及ADK版本
├── MyOperator.iml
```

----结束

IR 定义配置说明

表 8-6 IR 原型定义参数说明

列名称	含义	是否必选
Op	算子的Operator Type	是
Classify	算子相关参数的类别，包含： ● 输入：Input ● 动态输入：DYNAMIC_INPUT ● 输出：Output ● 动态输出：DYNAMIC_OUTPUT ● 属性：Attr	是
Name	算子参数的名称。	是

列名称	含义	是否必选
Type	算子参数的类型。 包含如下取值： tensor、int、bool、float、ListInt、ListFloat等	是
TypeRange	针对类型为tensor的参数，需要配置tensor支持的类型。 包含如下取值： fp16,fp32,double,int8,int16,int32,int64,uint8,uint16,uint32,uint64,bool等。 框架为MindSpore时，需要将tensor的类型值转换为MindSpore所支持的值： I8_Default,I16_Default,I32_Default,I64_Default,U8_Default,U16_Default,U32_Default,U64_Default,BOOL_Default等。	否
Required	是否必须输入，有如下取值： ● TRUE ● FALSE	是
Doc	对应参数的描述。	否
Attr_Default_value	属性的默认值。	否
Format	针对类型为tensor的参数，配置为tensor支持的数据排布格式。 包含如下取值： ND,NHWC,NCHW,HWCN,NC1HWC0,FRACTAL_Z等	否

8.4.2 设置 Python 库

算子开发之前，开发者需要参考本章节设置算子工程依赖Python库。

- **设置全局依赖的Python SDK**：**必选操作。**但仅需配置一次，对全部算子工程生效，后续创建算子工程时无需重复配置。
- **设置当前算子工程依赖的Python SDK**：**必选操作。**每次创建算子工程后，必须进行此步配置，仅对当前算子工程生效。
- **（可选）添加自定义Python SDK**：**可选操作。**用户需要添加自定义Python SDK时配置。

设置全局依赖的 Python SDK

步骤1 在工程界面中，单击菜单栏中的“File > Project Structure”，进入**Project Structure**设置页面。

步骤2 在左侧菜单栏中选择“Platform Settings > SDKs”，参考图8-6添加Python解释器。

图 8-6 添加全局 Python 库

须知

若 + 的下拉菜单中没有”Add Python SDK...“选项，请完成步骤10安装Python Community Edition插件。若没有网络权限请配置proxy代理。

说明

从Interpreter后面的 ... 选择解析器版本，需要选择python3.7.5版本。

System Interpreter表示使用本地Python解释器，若选用以下三种方式，您需要提前安装依赖并配置python镜像，安装依赖包括conda、pipenv等。

1. Virtualenv Environment：使用虚拟管理环境，在创建Python虚拟环境的目录下创建独立的Python库和解释器，创建好后，会在创建目录下重新根据当前用户的~/.pip/pip.conf文件中配置的PIP源通过PIP工具进行库的下载。
2. Conda Environment：选择Anaconda python解释器并利用conda创建虚拟环境。
3. Pipenv Environment：整合了virtualenv与pip，可同时创建虚拟环境并进行库管理功能。

步骤3 添加完成后，请确保算子开发相关Python库已存在，包括：te、topi等。

如下图示例所示：

图 8-7 查看 Python 库

步骤4 在**Project Structure**窗口中单击**Apply**。

步骤5 单击**OK**，关闭**Project Structure**窗口。

----结束

设置当前算子工程依赖的 Python SDK

若算子代码开发过程中，您自行引入了其他第三方python库或自定义python库，需要参考如下操作在工程中添加相应的python sdk。

步骤1 在工程界面中，单击菜单栏中的“File > Project Structure”，进入**Project Structure**设置页面。

步骤2 在左侧菜单栏中选择“Project Settings > Project”，进行Project设置。

在**Project SDK**中，下拉选择**设置全局依赖的Python SDK**中配置的Python SDK，并单击**Apply**，如**图8-8**所示。

图 8-8 选择 Project SDK

步骤3 在左侧菜单栏中选择“Project Settings > Modules”，进行Modules设置。

在**Dependencies**页签中，下拉选择**8.4.2 设置Python库**中配置的Python SDK，并单击**Apply**，如**图8-9**所示。

图 8-9 选择 Modules SDK

步骤4 单击**OK**，关闭**Project Structure**窗口。

此时，在工程底部菜单栏中会显示如**图8-10**所示提示。

图 8-10 更新界面

步骤5 更新成功后，Python代码可以实现自动解析，如**图8-11**所示。单击导入的Python模块代码，然后按**Ctrl+Q**，会出现相应的Tips。按住Ctrl建，单击还可进入对应的模块定义文件。

图 8-11 Python 代码自动解析示例

说明

若仍然无法自动解析，请关闭工程后重新打开。

----结束

（可选）添加自定义 Python SDK

若算子代码开发过程中，您自行引入了其他第三方python库或自定义python库，需要参考如下操作在工程中添加相应的python sdk。

步骤1 在工程界面中，单击菜单栏中的“File > Project Structure”，进入**Project Structure**设置页面。

步骤2 在左侧菜单栏中选择“Platform Settings > SDKs”，进行Python SDK设置。

步骤3 选择对应的python库，在Classpath栏中单击“+”，可添加自定义的python库，如图8-12所示。

需要注意的是：

- 在新建的Python库中，务必确保系统默认的算子开发相关Python库已存在，包括te、topi等。
- 由于“Platform Settings”页签中的Python库为全局配置，对所有算子工程生效。

图 8-12 添加用户自定义 Python 库

步骤4 单击**OK**，关闭**Project Structure**窗口。

----结束

8.5 TBE 算子开发（TensorFlow）

8.5.1 算子代码实现

通过调用TBE DSL接口，在算子工程下的“tbe/impl/add.py”文件中进行Add算子的实现，包括算子函数定义、算子入参校验、compute过程实现及调度与编译。

代码模板介绍

MindStudio在“tbe/impl/add.py”中生成代码模板。

```
# 导入依赖的Python模块
import te.lang.cce as tbe
from te import tvm
from te.platform.fusion_manager import fusion_manager


# 算子计算函数
@fusion_manager.register("add")
def add_compute(x, y, z, kernel_name="add"):
    """
    To do: Implement the operator by referring to the
           TBE Operator Development Guide.
    """

    res = tbe.XXX(x, y)
    return res


# 算子定义函数
def add(x, y, z, kernel_name="add"):
    """
    To do: Implement the operator by referring to the
           TBE Operator Development Guide.
    """
    # 输入参数占位
    data_x = tvm.placeholder(x.get("shape"), dtype=x.get("dtype"), name="data_x")
```

```
data_y = tvm.placeholder(y.get("shape"), dtype=y.get("dtype"), name="data_y")

# 调用算子计算函数
res = add_compute(data_x, data_y, z, kernel_name)

# 自动调度
with tvm.target.cce():
    schedule = tbe.auto_schedule(res)

# 编译
config = {"name": kernel_name,
          "tensor_list": [data_x, data_y, res]}
tbe.build(schedule, config)
```

- 引入算子开发时依赖的Python模块。
  - “te.lang.cce”：引入TBE支持的特定域语言接口，包括常见的运算vmuls、vadds、matmul等。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}-linux*/atc/python/site-packages/te/te/lang/cce**目录下的python函数。
  - “te.tvm”：引入TVM后端代码生成机制。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}-linux*/atc/python/site-packages/te/te/tvm**目录下的python函数，使用方法请参见**https://docs.tvm.ai/**。
  - “te.platform.fusion_manager.fusion_manager”：提供了实现算子的UB自动融合的接口。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}-linux*/atc/python/site-packages/te/te/platform/fusion_manager.py**文件中的fusion_manager函数的定义。
- 模板生成以***算子名称***_**compute**命名的计算函数声明。
  - 若创建算子工程时选择的“From Template”或者“From Tensorflow”，输入输出参数及属性会根据原型定义自动生成。
  - 若创建算子工程时选择“Empty”，默认生成的参数为一个输入与一个输出，不带属性。
- 模板生成以***算子名称***命名的定义函数的声明与部分实现，模板中提供的实现函数中的示例代码包含如下功能：

  获取输入tensor的shape与dtype，对输入参数进行校验，对输入tensor进行占位，调用算子的compute函数进行计算，调度与编译。

算子定义函数实现

开发者需要根据MindStudio生成的模板代码进行算子计算函数的实现。同时需要在算子定义函数中增加算子输入/输出/属性的校验代码；由于Add算子允许两个输入数据的shape不同，但算子计算接口**te.lang.cce.vadd( )**要求两输入shape相同，因此需要对算子两个输入的shape进行广播并对其进行校验有助于在算子编译阶段，提前发现问题。修改后代码如下所示：

须知

若您通过Windows上远程打开的MindStudio，复制以下代码时可能会出现无法复制的情况，解决方法请参见**15.8.1 Windows上远程打开MindStudio时，偶现复制的内容无法粘贴到编辑器窗口中**。

```
from __future__ import absolute_import
from functools import reduce
from te import tvm
from te.platform.fusion_manager import fusion_manager
import te.lang.cce as tbe
from te.utils import para_check
from te.utils import shape_util

SHAPE_SIZE_LIMIT = 2147483648

# 实现Add算子的计算逻辑
@fusion_manager.register("add")
def add_compute(input_x, input_y, output_z, kernel_name="add"):
    # 将shape转换为list
    shape_x = shape_util.shape_to_list(input_x.shape)
    shape_y = shape_util.shape_to_list(input_y.shape)

    # shape_max取shape_x与shape_y的每个维度的大值
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")
    shape_size = reduce(lambda x, y: x * y, shape_max[:])
    if shape_size > SHAPE_SIZE_LIMIT:
        raise RuntimeError("the shape is too large to calculate")

    # 将input_x的shape广播为shape_max
    input_x = tbe.broadcast(input_x, shape_max)
    input_y = tbe.broadcast(input_y, shape_max)

    # 执行input_x + input_y
    res = tbe.vadd(input_x, input_y)

    return res

# 算子定义函数
@para_check.check_op_params(para_check.REQUIRED_INPUT, para_check.REQUIRED_INPUT,
                            para_check.REQUIRED_OUTPUT, para_check.KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
    # 获取算子输入tensor的shape与dtype
    shape_x = input_x.get("shape")
    shape_y = input_y.get("shape")

    # 检验算子输入类型
    check_tuple = ("float16", "float32", "int32")
    input_data_type = input_x.get("dtype").lower()
    para_check.check_dtype(input_data_type, check_tuple, param_name="input_x")

    # shape_max取shape_x与shape_y的每个维度的最大值
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")

     # 如果shape的长度等于1，就直接赋值，如果shape的长度不等于1，做切片，将最后一个维度舍弃（按照内存排布，最后一个维度为1与没有最后一个维度的数据排布相同，例如2*3=2*3*1，将最后一个为1的维度舍弃，可提升后续的调度效率）
    if shape_x[-1] == 1 and shape_y[-1] == 1 and shape_max[-1] == 1:
        shape_x = shape_x if len(shape_x) == 1 else shape_x[:-1]
        shape_y = shape_y if len(shape_y) == 1 else shape_y[:-1]
        shape_max = shape_max if len(shape_max) == 1 else shape_max[:-1]

    # 使用TVM的placeholder接口对输入tensor进行占位，返回一个tensor对象
    data_x = tvm.placeholder(shape_x, name="data_1", dtype=input_data_type)
    data_y = tvm.placeholder(shape_y, name="data_2", dtype=input_data_type)

    # 调用compute实现函数
    res = add_compute(data_x, data_y, output_z, kernel_name)

    # 自动调度
    with tvm.target.cce():
```

```
    schedule = tbe.auto_schedule(res)
# 编译配置
config = {"name": kernel_name,
          "tensor_list": (data_x, data_y, res)}
tbe.build(schedule, config)
```

1. 算子定义函数声明包含算子输入信息、输出信息以及内核名称。

   ```
   def add(input_x, input_y, output_z, kernel_name="add"):
   ```

   - input_x, input_y：Add算子的两个输入tensor，每个tensor需要采用字典的形式进行定义，包含shape和dtype等信息。输入tensor的个数需要与**算子信息定义文件**“tbe/op_info_cfg/ai_core/add.ini”中的定义保持一致。
   - output_z：输出tensor，包含shape和dtype等信息，字典格式，此字段为预留位。

     输出tensor的个数需要与**算子信息定义文件**“tbe/op_info_cfg/ai_core/add.ini”中的定义保持一致。
   - kernel_name：算子在内核中的名称（即生成的二进制文件以及算子描述文件的名称），用户自定义，保持唯一，只能是大小写字母、数字、“_”的组合，且必须是字母或者“_”开头，长度小于或等于200个字符。
2. 算子输入校验和输出形状推导。

   Add算子需要对两个输入tensor的shape进行校验，且仅支持数据类型float16, float32, int32，此外Add算子允许两个输入shape不同，因此需要调用shape_util.broadcast_shapes()生成广播后的shape并对其进行校验。
3. 对输入tensor进行占位。

   调用TVM的placeholder接口分别对两个输入tensor进行占位，并分别返回一个tensor对象。

   须知

   5中的tensor_list的输入tensor需要是tvm.placeholder接口返回的tensor对象，所以此对象在后续计算过程实现中不能被替换。
4. 调用add_compute计算函数。

   ```
   res = add_compute(data_x, data_y, output_z, kernel_name)
   ```

   data_x、data_y为**3**生成的占位tensor对象。

   计算函数的实现请参见**算子Compute函数实现**。
5. 算子调度与编译实现。

   调度配置config中的tensor_list为：

   ```
   "tensor_list": (data_x, data_y, res)
   ```

   分别为两个输入tensor与一个输出tensor。

算子 Compute 函数实现

开发者需要根据算子计算逻辑自定义实现算子compute函数，Add算子的Compute函数实现如下：

```
@fusion_manager.register("add")
def add_compute(input_x, input_y, output_z, kernel_name="add"):
    # 将shape转换为list
    shape_x = shape_util.shape_to_list(input_x.shape)
    shape_y = shape_util.shape_to_list(input_y.shape)

    # shape_max取shape_x与shape_y的每个维度的大值
```

```
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")
    shape_size = reduce(lambda x, y: x * y, shape_max[:])
    if shape_size > SHAPE_SIZE_LIMIT:
        raise RuntimeError("the shape is too large to calculate")

    # 将input_x的shape广播为shape_max
    input_x = tbe.broadcast(input_x, shape_max)
    input_y = tbe.broadcast(input_y, shape_max)

    # 执行input_x + input_y
    res = tbe.vadd(input_x, input_y)

    return res
```

1. add_compute函数的声明如下所示。

   ```
   @fusion_manager.register("add")
   def add_compute(input_x, input_y, output_z, kernel_name="add")
   ```

   装饰器**@fusion_manager.register("add")**是DSL算子开发方式中必需的，其作用是整网运行时算子支持做UB自动融合，使得当前自定义算子可以在UB中根据UB融合规则自动与其他算子的计算进行组装。

   其中：

   – input_x，input_y：compute函数的入参，为在3中声明的输入tensor对应的placeholder，包含shape和dtype等信息。
   – output_z：为1中算子接口函数透传过来的dict类型。
   – kernel_name：算子在内核中的名称。

2. 进行Add算子的计算逻辑的实现。

   Add算子要求相加的两个tensor的shape相同，所以首先通过调用tbe.broadcast接口将两个输入tensor广播成相同的shape，然后调用tbe.vadd接口实现输入tensor的相加，并返回计算结果tensor。

算子编译验证

步骤1 在算子python文件最下方添加main函数调用该算子，通过MindStudio编译算子实现文件，用于单算子代码的简单语法校验，代码示例如下所示：

```
# 算子调用
if __name__ == '__main__':
    input_output_dict = {"shape": (5, 6, 7),"format": "ND","ori_shape": (5, 6, 7),"ori_format": "ND", "dtype":
"float16"}
    add(input_output_dict, input_output_dict, input_output_dict, kernel_name="add")
```

步骤2 右键单击“tbe/impl/*add.py*”，选择**Run‘*add*’**，编译算子。

如果编译没有报错，且在当前目录“tbe/impl”下生成kernel_meta文件夹，包括以下文件，则表示算子代码能够编译运行。

- 算子二进制文件*.o。
- 算子描述文件*.json：用于定义算子属性及运行时所需要的资源。

----结束

8.5.2 算子原型定义

进入“op_proto/”目录，编写IR实现文件“add.h”和“add.cpp”，将算子注册到算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。

add.h 实现

MindStudio已在add.h文件中生成了算子注册的代码模板文件，开发者可根据需要进行修改，Add算子的原型定义如下所示：

```
#ifndef GE_OPS_OP_PROTO_ADD_H_
#define GE_OPS_OP_PROTO_ADD_H_
#include "graph/operator_reg.h"
namespace ge {
REG_OP(Add)
    .INPUT(x1,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .INPUT(x2,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .OUTPUT(y,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .OP_END_FACTORY_REG(Add)
}

#endif //GE_OPS_OP_PROTO_ADD_H_
```

- REG_OP(Add)中的Add为算子注册到昇腾AI处理器中的Type，需要与**8.5.4 算子适配插件实现**中REGISTER_CUSTOM_OP("Add")中的算子类型保持一致。
- .INPUT与.OUTPUT分别为算子的输入、输出Tensor的名称与数据类型，输入输出的顺序需要与**8.5.1 算子代码实现**函数形参顺序以及**算子信息定义**中参数的顺序保持一致。

add.cpp 实现

开发者需要在add.cpp中实现InferShape与Verify方法。

- Verify函数，即如下代码示例中的IMPLEMT_VERIFIER(Add, AddVerify)函数，用于校验Add算子的两个输入的DataType是否一致。
- InferShape函数，即如下代码示例中的IMPLEMT_COMMON_INFERFUNC(AddInferShape)函数，用于推导出算子的输出张量描述，这样在网络运行时就可以为所有的张量静态分配内存，避免动态内存分配带来的开销。

add.cpp的实现代码如下所示：

```
#include "./add.h"          //IR注册头文件
#include <vector>         //可使用vector类模板并调用vector相关接口。
#include <string>         //C++标准库，可使用string类构造对象并调用string相关接口

namespace ge {
bool InferShapeAndTypeAdd(Operator& op,
    const string& inputName1,
    const string& inputName2, const string& outputName)
 {
    TensorDesc vOutputDesc = op.GetOutputDescByName(outputName.c_str());

    DataType inputDtype = op.GetInputDescByName(inputName1.c_str()).GetDataType();
    Format inputFormat = op.GetInputDescByName(inputName1.c_str()).GetFormat();
    // 针对shape维度大小进行交换
    ge::Shape shapeX = op.GetInputDescByName(inputName1.c_str()).GetShape();
    ge::Shape shapeY = op.GetInputDescByName(inputName2.c_str()).GetShape();
    std::vector<int64_t> dimsX = shapeX.GetDims();
    std::vector<int64_t> dimsY = shapeY.GetDims();
    if (dimsX.size() < dimsY.size()) {
```

```
        std::vector<int64_t> dimsTmp = dimsX;
        dimsX = dimsY;
        dimsY = dimsTmp;
    }

  // 对小的shape进行1补齐
  if (dimsX.size() != dimsY.size()) {
    int dec = dimsX.size() - dimsY.size();
    for (int i = 0; i < dec; i++) {
      dimsY.insert(dimsY.begin(), (int64_t)1);
    }
  }

    // 设置输出的shape维度
    std::vector<int64_t> dimVec;
    for (size_t i = 0; i < dimsX.size(); i++) {
        if ((dimsX[i] != dimsY[i]) && (dimsX[i] != 1) && (dimsY[i] != 1)) {
            return false;
        }

        int64_t dims = dimsX[i] > dimsY[i] ? dimsX[i] : dimsY[i];
        dimVec.push_back(dims);
    }
    ge::Shape outputShape = ge::Shape(dimVec);

    vOutputDesc.SetShape(outputShape);
    vOutputDesc.SetDataType(inputDtype);
    vOutputDesc.SetFormat(inputFormat);
    op.UpdateOutputDesc(outputName.c_str(), vOutputDesc);

    return true;
}

//----------------Add-------------------
IMPLEMT_VERIFIER(Add, AddVerify)
{
    if (op.GetInputDescByName("x1").GetDataType() != op.GetInputDescByName("x2").GetDataType()) {
        return GRAPH_FAILED;
    }
    return GRAPH_SUCCESS;
}


// Obtains the processing function of the output tensor description.
IMPLEMT_COMMON_INFERFUNC(AddInferShape)
{
    if(InferShapeAndTypeAdd(op, "x1", "x2", "y")) {
        return GRAPH_SUCCESS;
    }
    return GRAPH_FAILED;
}


//Registered inferfunction，第一个参数为算子的OpType
COMMON_INFER_FUNC_REG(Add, AddInferShape);


//Registered verify function，第一个参数为算子的OpType
VERIFY_FUNC_REG(Add, AddVerify);
//----------------Add-------------------
}
```

8.5.3 算子信息定义

需要通过配置算子信息文件，将算子的相关信息注册到算子信息库中。算子信息库主要体现算子在昇腾AI处理器上物理实现的限制，包括算子的输入输出dtype、format以及输入shape信息。网络运行时，FE会根据算子信息库中的算子信息做基本校验，判断

是否需要为算子插入合适的转换节点，并根据算子信息库中信息找到对应的算子实现文件进行编译，生成算子二进制文件进行执行。

进入“tbe/op_info_cfg/ai_core/*soc_version*”目录，配置算子信息库文件“add.ini”。

*soc_version*为当前昇腾AI处理器的版本，例如ascend310、ascend910等。

开发者需要基于MindStudio自动生成的add.ini文件进行修改，修改后的Add算子的算子信息定义如下所示。

```
[Add]
input0.name=x1
input0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
input0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
input0.shape=all
input0.paramType=required
input1.name=x2
input1.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
input1.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
input1.shape=all
input1.paramType=required
output0.name=y
output0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
output0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
output0.shape=all
output0.paramType=required
opFile.value=add
opInterface.value=add
```

参数说明请参见**表8-7**。下表仅列出常用的算子信息库配置项，其他配置项请参见

以下手册中的算子信息库定义章节。

- 《**TBE自定义算子开发指南(推理)**》
- 《**TBE自定义算子开发指南(训练)**》

表 8-7 Add 算子的算子信息定义说明

信息	Add算子配置	说明
[OpType]	[Add]	算子类型，以英文半角方括号，标识一个算子信息开始，根据**8.3 算子分析**，算子类型为Add。
input0.name	x1	Add算子的第一个输入tensor的名称，根据**8.3 算子分析**，第一个输入名称为x1，所以此处配置为x1，且需要跟**算子原型定义**中的名称保持一致。
input0.dtype	input0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32 input0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,N	定义输入tensor支持的数据类型与数据排布格式。 根据**8.3 算子分析**，Add算子的输入数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW、NC1HWC0、NHWC、ND。 注意：

input0.format	CHW,NC1HWC0,NHWC,ND	若算子输入支持多种规格，算子输入的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
input0.shape	all	定义输入tensor支持的形状。
input0.paramType	required	定义输入tensor的类型。 ● dynamic：表示该输入是动态个数，可能是1个，也可能是多个。 ● optional：表示该输入为可选，可以有1个，也可以不存在。 ● required：表示该输入有且仅有1个。 Add算子的input0为固定输入1个，此处配置为required。
input1.name	x2	因为Add算子有两个输入，生成的配置模板中只有一个输入，所以此处需要添加input1的相关配置。 此配置项代表Add算子的第二个输入tensor的名称，根据**8.3 算子分析**，第二个输入名称为x2，所以此处配置为x2，且需要跟**算子原型定义**中的名称保持一致。
input1.dtype	input1.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32 input1.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND	定义输入tensor支持的数据类型与数据排布格式。 根据**8.3 算子分析**，Add算子的输入数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW,NC1HWC0,NHWC,ND。 注意： 若算子输入支持多种规格，算子输入的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
input1.format		
input1.shape	all	定义输入tensor支持的形状。

版权所有 © 华为技术有限公司

input1.paramType	required	定义输入tensor的类型。 ● dynamic：表示该输入是动态个数，可能是1个，也可能是多个。 ● optional：表示该输入为可选，可以有1个，也可以不存在。 ● required：表示该输入有且仅有1个。 Add算子的input1为固定输入1个，此处配置为required。
output0.name	y	Add算子的输出tensor的名称，根据**8.3 算子分析**，算子的输出名称为y，所以此处配置为y，且需要跟**算子原型定义**中的名称保持一致。
output0.dtype output0.format	output0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32 output0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND	定义输出tensor支持的数据类型与数据排布格式 根据**8.3 算子分析**，Add算子的输出数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW,NC1HWC0,NHWC,ND。 注意： 若算子输出支持多种规格，算子输出的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
output0.shape	all	定义输出tensor支持的形状。
output0.paramType	required	定义输出tensor的类型。 ● dynamic：表示该输出是动态个数，可能是1个，也可能是多个。 ● optional：表示该输出为可选，可以有1个，也可以不存在。 ● required：表示该输出有且仅有1个。 Add算子的output0为固定输出1个，此处配置为required。
opFile.value	add	算子代码实现文件的名称。 因为算子实现代码的名称符合算子OpType到代码实现名称的转换规则，参见**步骤4**，所以此字段可不配置。

opInterface.value	add	算子实现文件中定义函数的名称。 因为算子定义函数的名称符合算子OpType到算子定义函数名称的转换规则，参见步骤4，所以此字段可不配置。

8.5.4 算子适配插件实现

开发者需要进行算子适配插件的开发，实现将Tensorflow网络中的算子进行解析并映射成昇腾AI处理器中的算子。MindStudio在“framework/tf_plugin/tensorflow_add_plugin.cpp”文件已自动生成了Add算子的插件代码，如图8-13所示。

图 8-13 插件代码实现模板

```
#include "register/register.h"

namespace domi {
REGISTER_CUSTOM_OP("Add")
    .FrameworkType(TENSORFLOW)
    .OriginOpType({ge::AscendString("Add")})
    .ParseParamsByOperatorFn(AutoMappingByOpFn)
    .ImplyType(ImplyType::TVM);
}  // namespace domi
```

- 包含头文件。
```
//包含该头文件，可使用算子注册类相关，调用算子注册相关的接口，为ADK中atc/include/register/register.h文件
#include "register/register.h"
```
- 进行插件注册。
```
namespace domi{
REGISTER_CUSTOM_OP("Add")
    .FrameworkType(TENSORFLOW)
    .OriginOpType({ge::AscendString("Add")})
    .ParseParamsByOperatorFn(AutoMappingFn)
    .ImplyType(ImplyType::TVM);    //需手动添加
}
```
  - REGISTER_CUSTOM_OP：算子注册到GE的算子类型，根据算子分析，算子类型为“Add”。
  - FrameworkType：TENSORFLOW，原始框架类型为TensorFlow。
  - OriginOpType：算子在TensorFlow框架中的类型。
  - ParseParamsByOperatorFn：用来注册解析模型的函数，使用AutoMappingFn函数自动实现解析。
  - ImplyType：指定算子的实现方式，ImplyType::TVM表示该算子是TBE算子。需要手动添加。

8.5.5 UT 测试

8.5.5.1 简介

MindStudio提供了基于gtest框架的新的UT测试方案，简化了开发者开发UT测试用例的复杂度。

UT（Unit Test：单元测试）是开发人员进行单算子运行验证的手段之一，主要目的是：

- 测试算子代码的正确性，验证输入输出结果与设计的一致性。
- UT侧重于保证算子程序能够跑通，选取的场景组合应能覆盖算子代码的所有分支（一般来说覆盖率要达到100%），从而降低不同场景下算子代码的编译失败率。

说明

测试类的详细定义可参见***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}*-*linux***/**toolkit/python/site-packages/op_test_frame/ut/op_ut.py**文件。

8.5.5.2 接口参考

8.5.5.2.1 OpUT

概述

OpUT为UT测试框架的基类，提供了测试用例定义及测试用例执行的接口，主要包含如下两个接口：

- add_case
- add_precision_case

OpUT 测试类定义

- 函数原型

  OpUT(op_type, op_module_name=None, op_func_name=None)
- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add（文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如BiasAdd算子自动生成module名称"impl.bias_add"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_case 接口

- 函数原型

  OpUT.add_case(support_soc=None, case=None)
- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - case：该参数为dict类型，示例如下：

```
{
  "params": [
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    },
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    }
  ],
  "case_name": "test_add_case_1",
  "expect": "success"
}
```

    该dict中key字段含义如下：

    - params：该字段在测试用例运行时透传给算子接口。
    - case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

      test_[op_type]_auto_case_name_[case_count]

      例如： test_Add_auto_case_name_1
    - expect：期望结果。默认为期望“success”，也可以是预期抛出的异常，例如RuntimeError。

add_precision_case 接口

- 函数原型

  OpUT.add_precision_case(support_soc=None, case)
- 功能说明

  添加算子编译+精度测试的用例。
- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - case：该参数为dict类型，示例如下：

```
{
  "params": [
    {
      "shape": (32, 64),
```

```
        "ori_shape": (32, 64),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16",
        "param_type": "input"
      },
      {
        "shape": (32, 64),
        "ori_shape": (32, 64),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16",
        "param_type": "output"
      }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add #一个函数
    "precision_standard": precision_info.PrecisionStandard(0.001, 0.001)  #可选字段
}
```

该dict中key字段含义如下：

- params：该字段在测试用例运行时传给算子接口。
- case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

  test_[op_type]_auto_case_name_[case_count]

  例如： test_Add_auto_case_name_1
- calc_expect_func：期望结果生成函数。
- precision_standard：自定义精度标准，若不配置此字段，按照如下默认精度与期望数据进行比对：
  - float16：双千分之一，即每个数据之间的误差不超过千分之一，误差超过千分之一的数据总和不超过总数据数的千分之一。
  - float32：双万分之一，即每个数据之间的误差不超过万分之一，误差超过万分之一的数据总和不超过总数据数的万分之一。

- 调用示例

  示例1：

```
from op_test_frame.ut import OpUT
# "ut_case" 为UT测试框架的关键字，不可修改
ut_case = OpUT("Add", "impl.add", "add")

def np_add(x1, x2, y):
    y = (x1.get("value") + x2.get("value"), )
    return y

ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
```

```
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add
})

if __name__ == '__main__':
    ut_case.run("Ascend910",None,"ca","/home/allan/Ascend/toolkit/tools/simulator")
```

说明：

ut_case.run：请参见**run接口**。

此样例未设置算子的输入数据值，测试框架将默认使用np.random.uniform(value_range, size=shape).astype(dtype)为每一个输入自动生成输入数据。

其中value_range使用默认值[0.1, 1.0]，shape和dtype的取值为“params”中的每一个输入。value_range也可以通过参数指定，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value_range": [2.0, 3.0]
}
```

也可以指定输入的值，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value": np.zeros((32, 64), np.float16)
}
```

示例2：

自定义精度标准示例如下：

```
from op_test_frame.common import precision_info
from op_test_frame.ut import OpUT

ut_case = OpUT("Add")
ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
```

```
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add,
    "precision_standard": precision_info.PrecisionStandard(0.1, 0.1) # 使用精度标准双十分之一，与期望数据进行比对
})
```

run 接口

- 函数原型

  OpUT.run(soc, case_name=None, simulator_mode=None, simulator_lib_path=None)

- 功能说明

  执行测试用例。

- 参数说明
  - soc：执行测试用例的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。
  - case_name：指定执行的case，配置为**add_case接口**及**add_precision_case接口**中的“case_name”
  - simulator_lib_path: 仿真库所在的路径。

    该路径结构如下所示：

    ```
    simulator_lib_path/
      Ascend910/
        lib/
          libpv_model.so
          ...
      Ascend310/
        lib/
          libpv_model.so
          ...
    ```

须知

使用MindStudio运行UT测试用例时，无需用户手工调用OpUT.run接口。

8.5.5.2.2 BroadcastOpUT

概述

BroadcastOpUT继承了OpUT，包含了OpUT的能力。

BroadcastOpUT主要供双输入单输出的Broadcast类型的算子进行测试用例的定义，例如Add、Mul等算子。BroadcastOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

BroadcastOpUT 测试类定义

- 函数原型

  BroadcastOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add（文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如BiasAdd算子自动生成module名称"impl.bias_add"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_broadcast_case 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case(self, soc, input_1_info, input_2_info, output_info=None,expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - input_1_info：算子的第一个输入的信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - input_2_info: 算子的第二个输入的信息，与input_1_info含义相同。
  - output_info：默认为None，不需要填写。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

  ```
  ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND"], ["float16", (32, 32), "ND"])

  ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND", (32, 32), "ND"], ["float16", (32, 32), "ND", (32, 32), "ND"])
  ```

```
# 期望异常的用例
ut_case.add_broadcast_case("all", ["float16", (31, 32), "ND"], ["float16", (32, 32), "ND"],
expect=RuntimeError)
```

add_broadcast_case_simple 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case_simple(self, soc, dtypes, shape1, shape2, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。此接口较add_broadcast_case更加简化。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape1: 算子的第一个输入的shape。
  - shape2：算子的第二个输入的shape。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_broadcast_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_broadcast_case_simple(["Ascend910", "Ascend310"], ["float16", "float32"], (32, 32), (32,
32))
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
```

```
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.5.5.2.3 ElementwiseOpUT

概述

ElementwiseOpUT继承了OpUT，包含了OpUT的能力。

ElementwiseOpUT主要供单输入单输出的Elementwise类型的算子进行测试用例的定义，例如Abs, Square等算子。

ElementwiseOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ElementwiseOpUT 测试类定义

- 函数原型

  ElementwiseOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add（文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如MaximumGrad算子自动生成module名称"impl.maximum_grad"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_elewise_case 接口

- 函数原型

  BroadcastOpUT.add_elewise_case(self, soc, param_info, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - param_info：算子的输入信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_elewise_case("Ascend910", ["float16", (32, 32), "ND"])
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})
```

add_elewise_case_simple 接口

- 函数原型

  BroadcastOpUT.add_elewise_case_simple(self, soc, dtypes, shape, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape：算子输入的shape。

- expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
- case_name: 默认为None，测试框架会自动生成测试用例的名称。

此接口较add_elewise_case接口，相当于将输入的所有format配置为“ND”。

示例：

```
ut_case.add_elewise_case_simple("Ascend910", ["float16", "float32"], [32, 32])
```

以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.5.5.2.4 ReduceOpUT

概述

ReduceOpUT继承了OpUT，包含了OpUT的能力。

ReduceOpUT主要供Reduce类型的算子进行测试用例的定义，例如ReduceSum, ReduceMean等算子。ReduceOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ReduceOpUT 测试类定义

- 函数原型

  ReduceOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据

op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如ReduceSum算子自动生成module名称"impl.reduce_sum"。

– op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

```
@check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
```

add_reduce_case 接口

- 函数原型

  BroadcastOpUT.add_reduce_case(self, soc, input_info, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明

  – soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}-linux***/**atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。

  – input_info：算子的输入信息，有两种形式：

    ▪ [dtype, shape, format, ori_shape, ori_format]

    ▪ [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。

  – axes: reduce的轴信息。

  – keep_dims: 是否保持reduce的轴为1。True：是；False：否。

  – expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。

  – case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_reduce_case("Ascend910", ["float16", (32, 32), "ND"], [0,], False)
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32,),
        "ori_shape": (32,),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, [0,], False]
})
```

add_reduce_case_simple 接口

- 函数原型

  BroadcastOpUT.add_reduce_case_simple(self, soc, dtypes, shape, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-*linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape：算子输入的shape。
  - axes: reduce的轴信息。
  - keep_dims: 是否保持reduce的轴为1。True：是；False：否。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_reduce_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_reduce_case_simple("Ascend910", ["float16", "float32"], [32, 32], [1,], True)
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 1),
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, [1,], True]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 1),
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
```

```
    }, [1,], True]
})
```

8.5.5.3 创建和运行 UT 测试用例

前提条件

需完成自定义算子的开发，包括算子实现代码、算子原型定义，详情可参见**8.5.1 算子代码实现**、**8.5.2 算子原型定义**、**8.5.3 算子信息定义**。

操作步骤

步骤1 创建UT测试用例。

1. 创建UT测试用例，有以下三种入口：

   右键单击算子工程根目录，选择“**New Cases > UT Case**”。

   若已经存在了算子的UT测试用例，可以右键单击“**testcases**”目录，或者“**testcases > ut**”目录，选择“**New Cases > UT Case**”，创建UT测试用例。

2. 在弹出的算子选择界面，选择需要创建UT测试用例的算子，单击**OK**，如下图所示。

说明

若已存在此算子的UT测试用例，系统会提示“testcases/ut/ops_test/xx already exists. Do you want to overwrite?”

您可以选择“Overwrite”或者“Cancel”。

创建完成后，会在算子工程根目录下生成testcases文件夹，目录结构如下所示：

```
├── MyOperator          //工程根目录
│   ├── testcases
│   │   ├── libs              // gtest框架，为第三方依赖，用户无需关注
│   │   ├── ut
│   │   │   ├── ops_test
│   │   │   │   ├── add
│   │   │   │   │   ├── CMakeLists.txt       //用于编译可执行文件
│   │   │   │   │   ├── test_add_impl.py      //算子实现代码的测试用例文件
│   │   │   │   │   ├── test_add_proto.cpp    //算子原型定义代码的测试用例文件
│   │   │   │   ├── CMakeLists.txt           //用于编译可执行文件
│   │   │   │   ├── test_main.cpp            //测试用例调用总入口
│   │   │   ├ CMakeLists.txt
```

步骤2 编写算子实现代码的UT Python测试用例。

在“testcases/ut/ops_test/add/test_add_impl.py”文件中，编写算子实现代码的UT Python测试用例，计算出算子执行结果，并取回结果和预期结果进行比较，来测试算子逻辑的正确性。

```
import sys
from op_test_frame.ut import BroadcastOpUT  # 导入UT测试类，可根据算子类型选择使用哪个测试类。
```

```
ut_case = BroadcastOpUT("Add")      # 实例化UT测试用例，ut_case为UT测试框架关键字，不可修改；Add为算子的Type

def calc_expect_func(input_x, input_y, output_z):    # 自定义实现生成期望数据的函数
    res = input_x["value"] + input_y["value"]
    return [res, ]    # 返回期望数据

# 添加测试用例
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "output"}],
    "calc_expect_func": calc_expect_func
})
# 若定义多个用例，定义多个ut_case.add_precision_case函数
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "output"}],
    "calc_expect_func": calc_expect_func
})
```

1. 首先导入UT测试类，用户可根据算子类型自行选择使用哪个UT测试类，详细可参见**8.5.5.2 接口参考**。
2. 实例化测试用例，OpUT的使用方法可参见**OpUT测试类定义**。
3. 用户自定义实现生成期望数据的函数。
4. 添加测试用例。

   可参见**8.5.5.2 接口参考**查看每个测试类接口的使用方法。

   若要与期望数据进行结果的比对，请使用**add_precision_case接口**。

步骤3 编写算子原型定义的UT C++测试用例。

在“testcases/ut/ops_test/add/test_add_proto.cpp”文件中，编写算子原型定义的UT C++测试用例，用于定义算子实例、更新算子输入输出并调用InferShapeAndType函数，最后验证InferShapeAndType函数执行过程及结果的正确性。

1. 导入gtest测试框架和算子IR定义的头文件。

   UT的C++用例采用的是gtest框架，所以需要导入gtest测试框架；算子原型定义在原型定义头文件中，所以需要导入原型定义的*.h文件。

```
//导入gtest框架
#include <gtest/gtest.h>
//导入基础的vector类库
#include <vector>
//导入算子的IR定义头文件
#include "add.h"
```

2. 定义测试类。

   UT的C++用例采用的是gtest框架，所以需要定义一个类来继承gtest的测试类。

```
#include <gtest/gtest.h>
#include <vector>
#include "add.h"

class AddTest : public testing::Test {
protected:
    static void SetUpTestCase() {
        std::cout << "add test SetUp" << std::endl;
}
```

```
    static void TearDownTestCase() {
        std::cout << "add test TearDown" << std::endl;
    }
};
```

测试类的名称可自定义，以“Test”为后缀。

3. 编写测试用例。

   每一个场景写一个测试用例函数，该用例中需要构造算子实例，包括算子名称、shape、数据类型。然后调用InferShapeAndType函数，并将推导出的shape、dtype与预期结果进行对比。

   示例如下：

```
TEST_F(AddTest, add_test_case_1) {
    // 定义算子实例及输入shape和type，以TensorDesc实例承载
    ge::op::Add add_op;  //Add为算子的Type，需要与原型定义的REG_OP(OpType)中的OpType保持一致。
    ge::TensorDesc tensorDesc;
    ge::Shape shape({2, 3, 4});
    tensorDesc.SetDataType(ge::DT_FLOAT16);
    tensorDesc.SetShape(shape);

    // 更新算子输入，输入的名称需要与原型定义*.h文件中的名称保持一致，例如:x1与x2分别为Add算子的两个输入
    add_op.UpdateInputDesc("x1", tensorDesc);
    add_op.UpdateInputDesc("x2", tensorDesc);
    // 调用InferShapeAndType函数，InferShapeAndType()接口为固定接口，用例执行时会自动调用算子原型定义中的shape推导函数
    auto ret = add_op.InferShapeAndType();
    // 验证调用过程是否成功
    EXPECT_EQ(ret, ge::GRAPH_SUCCESS);

    // 获取算子输出并比较shape和type，算子输出的名字需要与原型定义*.h文件中的名称保持一致，例如:算子的输出为y
    auto output_desc = add_op.GetOutputDesc("y");
    EXPECT_EQ(output_desc.GetDataType(), ge::DT_FLOAT16);
    std::vector<int64_t> expected_output_shape = {2, 3, 4};
    EXPECT_EQ(output_desc.GetShape().GetDims(), expected_output_shape);
}
```

   若不同输入的shape不同，请自行定义多个TensorDesc对象进行设置，例如：

```
    ge::op::Operator1 operator1_op;  //Operator1为算子的Type
    ge::TensorDesc tensorDesc1;
    ge::TensorDesc tensorDesc2;
    ge::Shape shape1({2, 3, 4});
    ge::Shape shape2({3, 4, 5});
    tensorDesc1.SetDataType(ge::DT_FLOAT16);
    tensorDesc1.SetShape(shape1);
    tensorDesc2.SetDataType(ge::DT_FLOAT16);
    tensorDesc2.SetShape(shape2);

    // 更新算子输入
    operator1_op.UpdateInputDesc("x1", tensorDesc1);
    operator1_op.UpdateInputDesc("x2", tensorDesc2);
```

步骤4 运行算子实现文件的UT测试用例。

开发人员可以执行当前工程中所有算子的UT测试用例，也可以执行单个算子的UT测试用例。

- 右键单击“testcases/ut/ops_test”文件夹，选择**Run Tbe Operator‘All’UT Impl with coverage**，执行整个文件夹下算子实现代码的测试用例。
- 右键单击“testcases/ut/ops_test/*算子名称*”文件夹，选择**Run Tbe Operator‘*算子名称*’UT Impl with coverage**，执行单个算子实现代码的测试用例。

第一次运行时会弹出运行配置页面，请参考配置，然后单击**Run**。后续如需修改运行配置，请参考**13.3.1 修改运行配置**。

表 8-8 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_impl。
SoC Version	下拉选择当前版本的昇腾AI处理器类型。
Target	运行环境。 – Simulator_Function：功能仿真环境。 – Simulator_TMModel：快速展示算子执行的调度流水线，不进行实际算子计算。 **当前功能仿支持Ascend 310和Ascend 910系列AI处理器。**
Operator Name	选择运行的测试用例。 – all表示运行所有用例。 – 其他表示运行某个算子下的测试用例。
Case Names	勾选需要运行的测试用例，即算子实现代码的UT Python测试用例。支持全选和全不选所有测试用例。

- 查看运行结果。
  a. 运行完成后，通过界面下方的“Run”日志打印窗口查看运行结果。
  b. 在“Run”窗口中单击index.html的URL（URL中的localhost为MindStudio安装服务器的IP，建议直接单击打开），查看UT测试用例的覆盖率结果，如图8-14所示。

  说明

  查看UT测试用例运行结果需要使用浏览器，如果未安装浏览器，请用户自行安装。

  图 8-14 查看 UT 用例覆盖率结果

  Coverage report: 100%

Module ↑	statements	missing	excluded	coverage
/home/[illegible]/MindStudio/samples/demo_projects/custom/tbe/impl/add.py	43	0	0	100%
Total	**43**	**0**	**0**	**100%**

  表示算子的UT用例覆盖率为100%，UT验证通过。

  如果出现"Page'http://***.html'requested without authorization, you can copy URL and open it in browser to trust it."提示，请参考**15.8.3 配置不受信任的网址访问浏览器**解决。

  c. 在html页面中单击对应算子，进入UT用例覆盖率详情页面，如图8-15所示，通过绿色和红色标签区分是否覆盖。

图 8-15 UT 覆盖率详情页面

d. 运行完成后，生成结果如下所示：

- 如果在配置运行信息选择的**Target**为**Simulator_TMModel**，还可以查看执行流水线，如下图所示。

说明

在芯片进行运算前，vector、cube、MTE1、MTE2、MTE3等单元会做初始化操作，对应在timeline显示上会出现数据还未搬运，各个单元就产生流水数据。

步骤5 运行算子原型定义的UT测试用例。

开发人员可以执行当前工程中所有算子的UT测试用例，也可以执行单个算子的UT测试用例。

- 右键单击“testcases/ut/ops_test”文件夹，选择**Run Tbe Operator‘All’UT Proto**，执行整个文件夹下算子原型定义代码的测试用例。
- 右键单击“testcases/ut/ops_test/*算子名称*”文件夹，选择**Run Tbe Operator‘*算子名称*’UT Proto**，执行单个算子原型定义代码的测试用例。

  第一次运行时会弹出运行配置页面，请参考配置，然后单击**Run**。后续如需修改运行配置，请参考**13.3.1 修改运行配置**。

表 8-9 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。

参数	说明
Test Type	选择ut_proto。
Operator Name	选择运行的测试用例。 – all表示运行所有用例。 – 其他表示运行某个算子下的测试用例。
Case Names	勾选需要运行的测试用例，即TEST_F中定义的用例。支持全选和全不选所有测试用例。

运行完成后，通过界面下方的日志打印窗口，查看运行结果。结果中展示此次一共运行几个用例，成功几个，失败几个，如下图所示。

----结束

8.5.6 算子工程编译

简介

将自定义算子工程编译生成自定义算子安装包custom_opp_*Target OS*_*Target Architecture*.run。

具体编译内容为：将算子插件实现文件、算子原型定义文件、算子信息定义文件分别编译成算子插件、算子原型库、算子信息库。编译过程示意如图8-16。

图 8-16 编译过程

编译操作

步骤1 在MindStudio工程界面，选中算子工程。

步骤2 单击顶部菜单栏的“ Build > Edit Build Configuration...”。

步骤3 进入编译配置页面，如下图所示。请参考表8-10进行编译配置。

图 8-17 编译配置界面

表 8-10 编译配置参数说明

参数	说明
Build Configuration	编译配置名称，默认为Build-Configuration
Build Mode	编译方式。 ● Remote Build：远端编译。 **须知** 远端编译需要g++版本为7.5.0。 ● Local Build：本地编译。 算子工程在MindStudio安装服务器进行编译，方便用户通过编译日志快速定位到MindStudio的实现代码所在位置，从而快速定位问题。此种方式下需要配置交叉编译环境，编译环境的配置请参考2.3.5 配置编译环境。

参数	说明
SSH Connection	Remote Build模式下显示该配置。下拉选择SSH配置信息，若未添加配置信息，请单击 ⊞ 添加。添加SSH配置信息的方法请参见13.1 SSH连接管理。
Target OS	Local Build模式下显示该配置。 ● 针对Ascend EP：选择昇腾AI处理器所在硬件环境的Host侧的操作系统。 ● 针对Ascend RC：选择板端环境的操作系统。
Target Architecture	Local Build模式下显示该配置。选择Target OS的操作系统架构。

步骤4 单击“Build”进行工程编译。

步骤5 在界面最下方的窗口查看编译结果，并在算子工程的cmake-build目录下生成自定义算子安装包**custom_opp_*Target OS*_*Target Architecture*.run**。

其中**Target OS** 与**Target Architecture**的取值参见步骤3中的配置。

----结束

8.5.7 算子部署（本地）

须知

- 执行8.5.9 ST测试操作时，会自动编译生成自定义算子安装包并将其部署到开发环境的opp目录下，若用户已参见8.5.9 ST测试进行了算子的ST测试，则此步骤可跳过；若需要单独部署自定义算子安装包，可参见此章节进行部署。
- 进行算子部署前，需要参见8.5.6 算子工程编译生成自定义算子安装包**custom_opp_*Targert OS*_*Target Architecture*.run**。

步骤1 在MindStudio工程界面菜单栏依次选择“Ascend > Deploy”。

弹出算子部署界面。

步骤2 在弹出的界面中选择“Deploy Locally”，并单击“Deploy”按钮。

在下方Output页签出现如下信息，代表自定义算子部署成功。

```
Output:  Output   Detail
2021-01-07 14:31:45  Uncompressing version:1.0  100%
2021-01-07 14:31:45  [runtime]  [INFO] copy uninstall sh success
2021-01-07 14:31:45  [ops_custom]upgrade framework
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops framework files ......
2021-01-07 14:31:45  [runtime]  copy new ops framework files ......
2021-01-07 14:31:45  [ops_custom]upgrade op proto
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops op_proto files ......
2021-01-07 14:31:45  [runtime]  copy new ops op_proto files ......
2021-01-07 14:31:45  [ops_custom]upgrade op impl
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops op_impl files ......
2021-01-07 14:31:45  [runtime]  copy new ops op_impl files ......
2021-01-07 14:31:45  [ops_custom]changemode...
2021-01-07 14:31:45  SUCCESS
Output  Log  6: TODO  4: Run  Profiling  Terminal  Pylint  Build
```

自定义算子包安装成功后，会将自定义算子部署在***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux/opp**目录下的对应文件夹中。

目录结构示例如下所示：

```
├── opp      //算子库目录
│   ├── op_impl
│     ├── built-in
│     ├── custom
│       ├── ai_core
│         ├── tbe
│           ├── config
│             ├── soc_version     //昇腾AI处理器版本
│               ├── aic-soc_version-ops-info.json     //自定义算子信息库文件
│           ├── custom_impl            //自定义算子实现代码文件
│             ├── add.py
│       ├── vector_core   //此目录预留，无需关注
│   ├── framework
│     ├── built-in
│     ├── custom
│       ├── caffe
│       ├── tensorflow          //存放tensorflow框架的自定义算子插件库
│         ├── libcust_tf_parsers.so
│         ├── npu_supported_ops.json   //Ascend 910场景下使用的文件，Ascend 310场景下无需关注
│   ├── op_proto
│     ├── built-in
│     ├── custom
│       ├── libcust_op_proto.so    //自定义算子原型库文件
```

----结束

8.5.8 算子部署（远程）

将自定义算子安装包**custom_opp_*Targert OS*_*Target Architecture*.run**部署到昇腾AI处理器所在硬件环境的算子库中，为后续算子在网络中运行构造必要条件。

步骤1 在MindStudio工程界面，选中算子工程。

步骤2 单击顶部菜单栏的“Ascend > Deploy”，进入算子打包部署界面。

请在**Deploy Remotely > SSH Connection**中下拉选择SSH配置信息，若未添加SSH配置信息，请单击 ⊞ 添加。

步骤3 配置环境变量

有两种方式：

- 在硬件设备的Host侧配置环境变量。

  MindStudio使用Host侧的运行用户在Host侧进行算子部署，进行算子部署执行前，需要在Host侧进行如下环境变量的配置。

  a. 以运行用户在Host侧的$HOME/.bashrc文件中配置如下环境变量。

  ```
  export ASCEND_OPP_PATH=/home/xxx/Ascend/opp
  ```

  /home/xxx/Ascend/为OPP组件（算子库）的安装路径，请根据实际情况配置。

  b. 执行命令使环境变量生效。

  source ~/.bashrc

- 在**Environment Variables**中添加环境变量。

  可以在**Environment Variables**中直接输入ASCEND_OPP_PATH=*/home/xxx/Ascend/opp*。

  */home/xxx/Ascend/*为OPP组件（算子库）的安装路径，请根据实际情况配置。

  也可以点击文本框后的图标，在弹出的对话框中填写。

  – 在Name中输入环境变量名称：ASCEND_OPP_PATH。
  – 在Value中输入环境变量值：*/home/xxx/Ascend/opp*。

图 8-18 算子打包部署界面

步骤4 选择算子部署的目标服务器，单击“Deploy”。

步骤5 算子部署过程即8.5.6 算子工程编译生成的自定义算子安装包的安装过程，部署完成后，算子被部署在Host侧算子库OPP对应文件夹中，默认路径为/usr/local/Ascend/opp/。

图 8-19 算子部署日志打印

```
2020-03-09 21:49:59  Verifying archive integrity... All good.
2020-03-09 21:49:59  Uncompressing version:1.0
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] copy uninstall sh success
2020-03-09 21:49:59  [ops_custom]upgrade framework
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops framework files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops framework files ......
2020-03-09 21:49:59  [ops_custom]upgrade op proto
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops op_proto files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops op_proto files ......
2020-03-09 21:49:59  [ops_custom]upgrade op impl
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops op_impl files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops op_impl files ......
2020-03-09 21:49:59  [ops_custom]changemode...
2020-03-09 21:49:59  SUCCESS
2020-03-09 21:49:59
2020-03-09 21:49:59  2020-03-10 09:49:59 - [INFO] Deploying the operator info to the remote host finished.
```

Host侧自定义算子部署完成后目录结构示例如下所示：

```
├── opp      //算子库目录
│   ├── op_impl
```

```
│       ├── built-in
│       ├── custom
│           ├── ai_core
│               ├── tbe
│                   ├── config
│                       ├── aic_ops_info.json     //自定义算子信息库文件
│                   ├── custom_impl               //自定义算子实现代码文件
│                       ├── add.py
│           ├── vector_core   //此目录预留，无需关注
│   ├── framework
│       ├── built-in
│       ├── custom
│           ├── caffe
│           ├── tensorflow          //存放tensorflow框架的自定义算子插件库
│               ├── libcust_tf_parsers.so
│   ├── op_proto
│       ├── built-in
│       ├── custom
│           ├── libcust_op_proto.so    //自定义算子原型库文件
```

----结束

8.5.9 ST 测试

概述

MindStudio提供了新的ST（System Test）测试框架，可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性和计算结果准确性，并生成运行测试报告，包括：

- 基于算子信息库生成算子测试用例定义文件。
- 基于算子测试用例定义文件生成不同shape、dtype的测试数据和基于AscendCL的测试用例。
- 编译算子工程并将算子部署到算子库，最后在硬件环境中执行测试用例，验证算子运行的正确性。
- 自动生成运行报表（st_report.json）功能，报表记录了测试用例信息及各阶段运行情况。
- 根据用户定义并配置的算子期望数据生成函数，回显期望算子输出和实际算子输出的对比测试结果，验证计算结果的准确性。

前提条件

- 完成自定义算子的开发，请参见**8.5.1 算子代码实现**、**8.5.2 算子原型定义**、**8.5.3 算子信息定义**、**8.5.4 算子适配插件实现**。
- MindStudio已连接硬件设备。

生成 ST 测试用例定义文件

步骤1 创建ST测试用例。

有以下三种入口：

- 右键单击算子工程根目录，选择“**New Cases > ST Case**”。
- 右键单击算子信息定义文件：***{工程名}*** **/tbe /op_info_cfg/ai_core/*{SoC Version}* /*xx*.ini**，选择“**New Cases > ST Case**”。
- 若已经存在了对应算子的ST Case，可以右键单击“**testcases**”目录，或者“**testcases > st**”目录，选择“**New Cases > ST Case**”，追加ST测试用例。

步骤2 在弹出的“Create ST Cases for an Operator”界面中选择需要创建ST测试用例的算子。

如下图所示：

Operator：下拉选择算子名称。

SoC Version：下拉选择昇腾AI处理器的类型。

- 若不勾选“Import operator info from a model”，单击“OK”后，会生成shape为空的算子测试用例定义文件，Design视图如下所示：

用户需要进行shape信息的配置，用于生成测试数据及测试用例，您也可以根据需要进行其他字段的配置，每个字段的详细要求可参见**算子测试用例定义文件参数解释**。

- 若用户勾选“Import operator info from a model”，选择包含算子的Tensorflow模型文件（*.pb）后，界面会显示获取到的模型文件的首层shape信息。

用户也可以在“Input Nodes Shape”中修改首层输入的shape信息。

单击“OK”后，工具会自动根据首层shape信息dump出选择算子的shape信息，生成对应的算子测试用例定义文件，测试用例定义文件有Text和Design两种视图，Design视图如下所示：

此文件用于生成测试数据及测试用例，您可以参见**算子测试用例定义文件参数解释**进行相关字段的修改。

步骤3 若要将算子与标杆数据对比，需要定义并配置算子期望数据生成函数。

1. 自定义实现算子期望数据生成函数。

   算子期望数据生成函数是用TensorFlow或Caffe等框架实现的与自定义算子功能相同的函数，其可以在CPU上运行并生成标杆数据。标杆数据用来与自定义算子生成数据进行对比，根据对比结果确定自定义算子精度。算子期望数据生成函数用python语言实现，在一个python文件中可以实现多个算子期望数据生成函数。该函数的输入、输出、属性与自定义算子的输入、输出、属性的Format、Type、Shape保持一致。

2. 配置算子测试用例定义文件。

   在算子测试用例文件中配置算子期望数据生成函数。有两种场景，分别为Design视图和Text视图下进行配置。

   – 若在Design视图下，配置方法为在Expected Result Verification下的Script Path中选择算子期望数据生成函数的python文件所在路径。在Script Function中输入算子期望数据生成函数的函数名。

说明

Script Function文本框中可以输入算子期望数据生成函数的函数名，也可以为空。

- 当输入算子期望数据生成函数的函数名时，ST测试时会使用该函数名对应的函数生成标杆数据。
- 当输入为空时，会自动匹配Script Path路径下的python文件中与自定义算子同名的函数，用于ST测试时生成标杆数据。若无同名函数，则会提示匹配失败。

   – 若在Text视图下，增加"calc_expect_func_file"参数，其值为算子期望数据生成函数对应的文件路径及算子函数名，如：

```
"calc_expect_func_file": "/home/teste/test_*.py:function",  //配置期望算子文件
```

   其中，/home/teste/test_*.py为算子期望数据生成函数的实现文件，function为对应的函数名称。算子期望数据生成函数文件路径和函数名称用冒号隔开。

步骤4 修改Case信息后，单击“Save”，修改会保存到算子测试用例定义文件。

算子测试用例定义文件存储目录为算子工程根目录下的“testcases/st/*OpType*/*{SoC Version}*”文件夹下，命名为*OpType*_case_*timestamp*.json

说明

请勿更改算子测试用例定义文件命名格式和将其他文件的以该命名格式保存在算子工程根目录下的“testcases/st/*OpType*/*{SoC Version}*”文件夹下。否则会导致文件解析错误。

----结束

运行 ST 测试用例

步骤1 右键单击**生成ST测试用例定义文件**中生成的ST测试用例定义文件（路径：“testcases > st > add > *{SoC Version}* > xxxx.json”），选择 "Run The Operator 'xxx' ST Case"。

表 8-11 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择st_cases 。

参数	说明
Execute Mode	● Remote Execute 远程执行测试 ● Local Execute 本地执行测试
SSH Connection	当Execute Mode选择Remote Execute时，下拉选择SSH配置信息，若未添加配置信息，请单击 ⊞ 添加。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。
Environment Variables	● 在文本框中添加环境变量 *PATH_1=路径1；PATH_2=路径2* 多个环境变量用英文分号隔开。 ● 也可以点击文本框后的图标，在弹出的对话框中填写。 – 在Name中输入环境变量名称：PATH_1。 – 在Value中输入环境变量值：*路径1*。 勾选Instead system environment variables可以显示系统环境变量。
Test Case Id	选择需要运行的算子。
SoC Version	选择处理器版本。
Executable File Name	下拉选择需要执行的测试用例定义文件。
Target OS	针对Ascend EP：选择昇腾AI处理器所在硬件环境的Host侧的操作系统。 针对Ascend RC：选择板端环境的操作系统。
Target Architecture	选择Target OS的操作系统架构。
Case Names	选择运行的Case Name。 **说明**：默认全选所有用例，可以去除勾选部分不需要运行的用例。

步骤2 Host侧运行用户设置。

Host侧运行用户需在SSH Connection中添加，并且该用户必须为HwHiAiUser属组。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。

步骤3 在运行环境上有两种方式配置相关组件的环境变量。

- 在远程设备上配置环境变量。
  - 针对Ascend EP，需要在硬件设备的Host侧配置安装组件路径的环境变量。

    以Host侧运行用户在~/.bashrc文件中配置acllib或fwkacllib、driver组件的安装路径。

    i. 打开运行用户下的.bashrc文件。

    vi ~/.bashrc

    ii. 在文件最后添加如下信息。

    export ASCEND_DRIVER_PATH=*/usr/local/Ascend/driver*

export ASCEND_HOME=*/usr/local/Ascend*/ascend-toolkit/latest

export ASCEND_AICPU_PATH=${ASCEND_HOME}/*<target architecture>*

若远程设备为推理环境：

export LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64:${ASCEND_HOME}/acllib/lib64:$LD_LIBRARY_PATH

若远程设备为训练环境：

export

LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64/driver:${ASCEND_DRIVER_PATH}/lib64/common:${ASCEND_HOME}/fwkacllib/lib64:$LD_LIBRARY_PATH

若已存在如上所述环境变量，请确认为当前运行环境实际安装组件所在路径。

<target architecture>为操作系统架构，例如：x86_64-linux或arm64-linux。

iii. 保存退出。

:wq

iv. 使环境变量生效。

source ~/.bashrc

- 在本地配置环境变量

  在**步骤1**中运行的窗口中Environment Variables参数进行环境变量配置。

  – 在文本框中添加环境变量

  ASCEND_DRIVER_PATH=*/usr/local/Ascend/driver;*
  ASCEND_HOME=*/usr/local/Ascend/ascend-toolkit/latest;*
  ASCEND_AICPU_PATH=${ASCEND_HOME}/*<target architecture>*;
  若远程设备为推理环境：
  LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64:${ASCEND_HOME}/acllib/lib64:$LD_LIBRARY_PATH;
  若远程设备为训练环境：
  LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64/driver:${ASCEND_DRIVER_PATH}/lib64/common:${ASCEND_HOME}/fwkacllib/lib64:$LD_LIBRARY_PATH;

  若已存在如上所述环境变量，请确认为当前运行环境实际安装组件所在路径。

  – 也可以点击文本框后的图标，在弹出的对话框中填写。

  分别在Name中输入环境变量名称，在Value中输入环境变量值。

步骤4 单击“**Run**”。

MindStudio会根据算子测试用例定义文件在“*算子工程根目录*/testcases/st/out/*OpType*”下生成测试数据和测试代码，并编译出可执行文件，在指定的硬件设备上执行测试用例，并将执行结果和与标杆数据对比报告打印到Output窗口中，同时在“*算子工程根目录*/testcases/st/out/*OpType*”下生成st_report.json文件。st_report.json文件信息含义请参见**表8-12**。

表 8-12 st_report.json 报表主要字段及含义

字段				说明
run_cmd	-	-	-	命令行命令。

字段				说明
report_list	-	-	-	报告列表，该列表中可包含多个测试用例的报告。
	trace_detail	-	-	运行细节。
		st_case_info	-	测试信息。
			expect_data_path	期望计算结果路径。
			case_name	测试用例名称。
			input_data_path	输入数据路径。
			planned_output_data_paths	实际计算结果输出路径。
			op_params	算子参数信息。
		stage_result	-	运行各阶段结果信息。
	case_name	-	-	测试名称。
	status	-	-	测试结果信息。

----结束

其他功能扩展

- 支持Design和Text界面视图的切换

  双击“testcases/st/*OpType*/*{SoC Version}*”文件夹下的算子测试用例定义文件或者右键该文件选择"Design Testcase *OpType*_case_*timestamp*.json"，会打开下图界面，该界面中的Design和Text均为json文件的一个视图。

  在Design视图中的更改，若该视图不存在错误配置信息，切换到Text视图时，更改会自动同步到Text视图中；同理，在Text视图中的更改，切换到Design视图时，更改也会同步到Design视图中，如下图所示：

- 新增测试用例

  在算子测试用例定义界面单击“Add”按钮，弹出如下图所示弹框，提示用户输入“Case Name”。

"Case Name"为由数字、字母、下划线组成的字符串。

版权所有 © 华为技术有限公司

单击“OK”后，会在Design Cases页面的尾部新增创建好的Case，新增Case的各个字段值为空，需要用户进行配置，配置规则可参见**算子测试用例定义文件参数解释**。

- 删除测试用例

  在算子测试用例定义界面，单击相应Case右侧的图标，即可删除该Case。

- 选择单个或多个测试用例运行

  在算子测试用例定义界面，勾选其中一个或多个Case，单击“Run”，即可运行选中的Case。

算子测试用例定义文件参数解释

表 8-13 算子测试用例定义 json 文件

参数		说明
Test_xxx_xx	-	必选。string类型。 测试用例的名称。

参数		说明
Input[xx]	-	必选。 算子的输入。 **须知** 一个算子所有Input中参数取值的个数都要一致，否则测试用例生成会失败。 例如：Input[01]的format支持的类型个数2，则Input[02]的format支持的类型个数也需要为2。 同理，所有Input[xx]中的type、shape、data_distribute和value_range的取值个数也需要保持一致。
-	Format	必选。 String或者String的一维数组。 输入tensor数据的排布格式，不允许为空。 支持如下数据排布格式： ● NCHW ● NHWC ● ND：表示支持任意格式。 ● NC1HWC0：华为自研的5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。 ● FRACTAL_Z：卷积的权重的格式。 ● FRACTAL_NZ：华为自研的分形格式，在cube单元计算时，输出矩阵的数据格式为NW1H1H0W0。整个矩阵被分为（H1*W1）个分形，按照column major排布，形状如N字形；每个分形内部有（H0*W0）个元素，按照row major排布，形状如z字形。考虑到数据排布格式，将NW1H1H0W0数据格式称为Nz格式。其中，H0,W0表示一个分形的大小，示意图如下所示：

参数		说明
-	Type	必选。 String或者String的一维数组。 输入数据支持的数据类型。 ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16 ● float
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状。
-	ValueRange	必选。 int类型或者float类型，一维或者二维数组。 取值范围，不能为空。 为[min_value, max_value]且min_value <=max_value。

参数		说明
-	DataDistribute	必选。 String或者String的一维数组。 使用哪种数据分布方式生成测试数据，支持的分布方式有： ● uniform：返回均匀分布随机值 ● normal：返回正态分布（高斯分布）随机值 ● beta：返回Beta分布随机值 ● laplace：返回拉普拉斯分布随机值 ● triangular：返回三角形分布随机值 ● relu：返回均匀分布+Relu激活后的随机值 ● sigmoid：返回均匀分布 + sigmoid激活后的随机值 ● softmax：返回均匀分布 + softmax激活后的随机值 ● tanh：返回均匀分布 + tanh激活后的随机值
Output[xx]	-	必选。 算子输出。 **须知** Output中参数取值的个数都要与Input一致，否则测试用例生成会失败。 例如：Input的format支持的类型个数2，则Output的format支持的类型个数也需要为2。

版权所有 © 华为技术有限公司

参数		说明
-	Format	必选。 String或者String的一维数组。 输出tensor数据的排布格式，不允许为空。 支持如下数据排布格式： ● NCHW ● NHWC ● ND：表示支持任意格式。 ● NC1HWC0：华为自研的5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。 ● FRACTAL_Z：卷积的权重的格式。 ● FRACTAL_NZ：华为自研的分形格式，在cube单元计算时，输出矩阵的数据格式为NW1H1H0W0。整个矩阵被分为（H1*W1）个分形，按照column major排布，形状如N字形；每个分形内部有（H0*W0）个元素，按照row major排布，形状如z字形。考虑到数据排布格式，将NW1H1H0W0数据格式称为Nz格式。其中，H0,W0表示一个分形的大小，示意图如下所示：

版权所有 © 华为技术有限公司

参数		说明
-	Type	必选。 String或者String的一维数组。 输出数据支持的数据类型： ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16 ● float
-	Shape	必选。 int类型，一维或者二维数组。 输出tensor支持的形状。
Attr	-	可选。
-	Name	若配置attr，则为必选。 string类型。 属性的名称，不为空。
-	Type	若配置attr，则为必选属性支持的类型。 string类型。 输出数据支持的数据类型： ● bool ● int ● float ● string ● list_bool ● list_int ● list_float ● list_string ● list_list_int

参数		说明
-	Value	若配置attr，则为必选，且不允许为null。 string类型。 属性值，根据type的不同，属性值不同。 ● bool: true/false ● int: 10 ● float: 1.0 ● string: “NCHW” ● list_bool: [false, true] ● list_int: [1, 224, 224, 3] ● list_float: [1.0, 0.0] ● list_string: [“str1”, “str2”] ● list_list_int: [[1, 3, 5, 7], [2, 4, 6, 8]]
Expected Result Verification	-	可选。
-	Script Path	算子期望数据生成函数路径。
-	Script Function	算子期望数据生成函数名称。

8.6 TBE 算子开发（PyTorch）

8.6.1 算子代码实现

通过调用TBE DSL接口，在算子工程下的“tbe/impl/add.py”文件中进行Add算子的实现，包括算子函数定义、算子入参校验、compute过程实现及调度与编译。

代码模板介绍

MindStudio在“tbe/impl/add.py”中生成代码模板。

```
# 导入依赖的Python模块
import te.lang.cce as tbe
from te import tvm
from te.platform.fusion_manager import fusion_manager


# 算子计算函数
@fusion_manager.register("add")
def add_compute(x, y, z, kernel_name="add"):
    """
    To do: Implement the operator by referring to the
           TBE Operator Development Guide.
    """

    res = tbe.XXX(x, y)
```

```
    return res


# 算子定义函数
def add(x, y, z, kernel_name="add"):
    """
    To do: Implement the operator by referring to the
           TBE Operator Development Guide.
    """
    # 输入参数占位
    data_x = tvm.placeholder(x.get("shape"), dtype=x.get("dtype"), name="data_x")
    data_y = tvm.placeholder(y.get("shape"), dtype=y.get("dtype"), name="data_y")

    # 调用算子计算函数
    res = add_compute(data_x, data_y, z, kernel_name)

    # 自动调度
    with tvm.target.cce():
        schedule = tbe.auto_schedule(res)

    # 编译
    config = {"name": kernel_name,
              "tensor_list": [data_x, data_y, res]}
    tbe.build(schedule, config)
```

- 引入算子开发时依赖的Python模块。
  - “te.lang.cce”：引入TBE支持的特定域语言接口，包括常见的运算vmuls、vadds、matmul等。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*|*{arch}-linux*/atc/python/site-packages/te/te/lang/cce**目录下的python函数。
  - “te.tvm”：引入TVM后端代码生成机制。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*|*{arch}-linux*/atc/python/site-packages/te/te/tvm**目录下的python函数，使用方法请参见**https://docs.tvm.ai/**。
  - “te.platform.fusion_manager.fusion_manager”：提供了实现算子的UB自动融合的接口。

    具体的接口定义可查看***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*|*{arch}-linux*/atc/python/site-packages/te/te/platform/fusion_manager.py**文件中的fusion_manager函数的定义。
- 模板生成以***算子名称*_compute**命名的计算函数声明。
  - 若创建算子工程时选择的“From Template”或者“From Tensorflow”，输入输出参数及属性会根据原型定义自动生成。
  - 若创建算子工程时选择“Empty”，默认生成的参数为一个输入与一个输出，不带属性。
- 模板生成以***算子名称***命名的定义函数的声明与部分实现，模板中提供的实现函数中的示例代码包含如下功能：

  获取输入tensor的shape与dtype，对输入参数进行校验，对输入tensor进行占位，调用算子的compute函数进行计算，调度与编译。

算子定义函数实现

开发者需要根据MindStudio生成的模板代码进行算子计算函数的实现。同时需要在算子定义函数中增加算子输入/输出/属性的校验代码；由于Add算子允许两个输入数据的shape不同，但算子计算接口**te.lang.cce.vadd( )**要求两输入shape相同，因此需要对

算子两个输入的shape进行广播并对其进行校验有助于在算子编译阶段，提前发现问题。修改后代码如下所示：

须知

若您通过Windows上远程打开的MindStudio，复制以下代码时可能会出现无法复制的情况，解决方法请参见**15.8.1 Windows上远程打开MindStudio时，偶现复制的内容无法粘贴到编辑器窗口中**。

```
from __future__ import absolute_import
from functools import reduce
from te import tvm
from te.platform.fusion_manager import fusion_manager
import te.lang.cce as tbe
from te.utils import para_check
from te.utils import shape_util

SHAPE_SIZE_LIMIT = 2147483648

# 实现Add算子的计算逻辑
@fusion_manager.register("add")
def add_compute(input_x, input_y, output_z, kernel_name="add"):
    # 将shape转换为list
    shape_x = shape_util.shape_to_list(input_x.shape)
    shape_y = shape_util.shape_to_list(input_y.shape)

    # shape_max取shape_x与shape_y的每个维度的大值
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")
    shape_size = reduce(lambda x, y: x * y, shape_max[:])
    if shape_size > SHAPE_SIZE_LIMIT:
        raise RuntimeError("the shape is too large to calculate")

    # 将input_x的shape广播为shape_max
    input_x = tbe.broadcast(input_x, shape_max)
    input_y = tbe.broadcast(input_y, shape_max)

    # 执行input_x + input_y
    res = tbe.vadd(input_x, input_y)

    return res

# 算子定义函数
@para_check.check_op_params(para_check.REQUIRED_INPUT, para_check.REQUIRED_INPUT,
                            para_check.REQUIRED_OUTPUT, para_check.KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
    # 获取算子输入tensor的shape与dtype
    shape_x = input_x.get("shape")
    shape_y = input_y.get("shape")

    # 检验算子输入类型
    check_tuple = ("float16", "float32", "int32")
    input_data_type = input_x.get("dtype").lower()
    para_check.check_dtype(input_data_type, check_tuple, param_name="input_x")

    # shape_max取shape_x与shape_y的每个维度的最大值
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")

     # 如果shape的长度等于1，就直接赋值，如果shape的长度不等于1，做切片，将最后一个维度舍弃（按照内存排布，最后一个维度为1与没有最后一个维度的数据排布相同，例如2*3=2*3*1，将最后一个为1的维度舍弃，可提升后续的调度效率）
    if shape_x[-1] == 1 and shape_y[-1] == 1 and shape_max[-1] == 1:
```

```
    shape_x = shape_x if len(shape_x) == 1 else shape_x[:-1]
    shape_y = shape_y if len(shape_y) == 1 else shape_y[:-1]
    shape_max = shape_max if len(shape_max) == 1 else shape_max[:-1]

# 使用TVM的placeholder接口对输入tensor进行占位，返回一个tensor对象
data_x = tvm.placeholder(shape_x, name="data_1", dtype=input_data_type)
data_y = tvm.placeholder(shape_y, name="data_2", dtype=input_data_type)

# 调用compute实现函数
res = add_compute(data_x, data_y, output_z, kernel_name)

# 自动调度
with tvm.target.cce():
    schedule = tbe.auto_schedule(res)
# 编译配置
config = {"name": kernel_name,
          "tensor_list": (data_x, data_y, res)}
tbe.build(schedule, config)
```

1. 算子定义函数声明包含算子输入信息、输出信息以及内核名称。

   ```
   def add(input_x, input_y, output_z, kernel_name="add"):
   ```

   - input_x, input_y：Add算子的两个输入tensor，每个tensor需要采用字典的形式进行定义，包含shape和dtype等信息。输入tensor的个数需要与**算子信息定义文件**“tbe/op_info_cfg/ai_core/add.ini”中的定义保持一致。
   - output_z：输出tensor，包含shape和dtype等信息，字典格式，此字段为预留位。

     输出tensor的个数需要与**算子信息定义文件**“tbe/op_info_cfg/ai_core/add.ini”中的定义保持一致。
   - kernel_name：算子在内核中的名称（即生成的二进制文件以及算子描述文件的名称），用户自定义，保持唯一，只能是大小写字母、数字、“_”的组合，且必须是字母或者“_”开头，长度小于或等于200个字符。
2. 算子输入校验和输出形状推导。

   Add算子需要对两个输入tensor的shape进行校验，且仅支持数据类型float16, float32, int32，此外Add算子允许两个输入shape不同，因此需要调用shape_util.broadcast_shapes()生成广播后的shape并对其进行校验。
3. 对输入tensor进行占位。

   调用TVM的placeholder接口分别对两个输入tensor进行占位，并分别返回一个tensor对象。

   须知

   5中的tensor_list的输入tensor需要是tvm.placeholder接口返回的tensor对象，所以此对象在后续计算过程实现中不能被替换。
4. 调用add_compute计算函数。

   ```
   res = add_compute(data_x, data_y, output_z, kernel_name)
   ```

   data_x、data_y为**3**生成的占位tensor对象。

   计算函数的实现请参见**算子Compute函数实现**。
5. 算子调度与编译实现。

   调度配置config中的tensor_list为：

   ```
   "tensor_list": (data_x, data_y, res)
   ```

   分别为两个输入tensor与一个输出tensor。

算子 Compute 函数实现

开发者需要根据算子计算逻辑自定义实现算子compute函数，Add算子的Compute函数实现如下：

```
@fusion_manager.register("add")
def add_compute(input_x, input_y, output_z, kernel_name="add"):
    # 将shape转换为list
    shape_x = shape_util.shape_to_list(input_x.shape)
    shape_y = shape_util.shape_to_list(input_y.shape)

    # shape_max取shape_x与shape_y的每个维度的大值
    shape_x, shape_y, shape_max = shape_util.broadcast_shapes(shape_x, shape_y,
                                                              param_name_input1="input_x",
                                                              param_name_input2="input_y")
    shape_size = reduce(lambda x, y: x * y, shape_max[:])
    if shape_size > SHAPE_SIZE_LIMIT:
        raise RuntimeError("the shape is too large to calculate")

    # 将input_x的shape广播为shape_max
    input_x = tbe.broadcast(input_x, shape_max)
    input_y = tbe.broadcast(input_y, shape_max)

    # 执行input_x + input_y
    res = tbe.vadd(input_x, input_y)

    return res
```

1. add_compute函数的声明如下所示。

   ```
   @fusion_manager.register("add")
   def add_compute(input_x, input_y, output_z, kernel_name="add")
   ```

   装饰器**@fusion_manager.register("add")**是DSL算子开发方式中必需的，其作用是整网运行时算子支持做UB自动融合，使得当前自定义算子可以在UB中根据UB融合规则自动与其他算子的计算进行组装。

   其中：

   - input_x，input_y：compute函数的入参，为在**3**中声明的输入tensor对应的placeholder，包含shape和dtype等信息。
   - output_z：为**1**中算子接口函数透传过来的dict类型。
   - kernel_name：算子在内核中的名称。

2. 进行Add算子的计算逻辑的实现。

   Add算子要求相加的两个tensor的shape相同，所以首先通过调用tbe.broadcast接口将两个输入tensor广播成相同的shape，然后调用tbe.vadd接口实现输入tensor的相加，并返回计算结果tensor。

算子编译验证

步骤1 在算子python文件最下方添加main函数调用该算子，通过MindStudio编译算子实现文件，用于单算子代码的简单语法校验，代码示例如下所示：

```
# 算子调用
if __name__ == '__main__':
    input_output_dict = {"shape": (5, 6, 7),"format": "ND","ori_shape": (5, 6, 7),"ori_format": "ND", "dtype":
"float16"}
    add(input_output_dict, input_output_dict, input_output_dict, kernel_name="add")
```

步骤2 右键单击“tbe/impl/*add.py*”，选择**Run‘*add*’**，编译算子。

如果编译没有报错，且在当前目录“tbe/impl”下生成kernel_meta文件夹，包括以下文件，则表示算子代码能够编译运行。

- 算子二进制文件*.o。

- 算子描述文件*.json：用于定义算子属性及运行时所需要的资源。

----结束

8.6.2 算子原型定义

进入“op_proto/”目录，编写IR实现文件“add.h”和“add.cpp”，将算子注册到算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。

add.h 实现

MindStudio已在add.h文件中生成了算子注册的代码模板文件，开发者可根据需要进行修改，Add算子的原型定义如下所示：

```
#ifndef GE_OPS_OP_PROTO_ADD_H_
#define GE_OPS_OP_PROTO_ADD_H_
#include "graph/operator_reg.h"
namespace ge {
REG_OP(Add)
    .INPUT(x1,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .INPUT(x2,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .OUTPUT(y,
        TensorType({DT_FLOAT, DT_INT32, DT_INT64, DT_FLOAT16, DT_INT16, DT_INT8, DT_UINT8,
DT_DOUBLE, DT_COMPLEX128,
            DT_COMPLEX64, DT_STRING}))
    .OP_END_FACTORY_REG(Add)
}

#endif //GE_OPS_OP_PROTO_ADD_H_
```

add.cpp 实现

开发者需要在add.cpp中实现InferShape与Verify方法。

- Verify函数，即如下代码示例中的IMPLEMT_VERIFIER(Add, AddVerify)函数，用于校验Add算子的两个输入的DataType是否一致。
- InferShape函数，即如下代码示例中的IMPLEMT_COMMON_INFERFUNC(AddInferShape)函数，用于推导出算子的输出张量描述，这样在网络运行时就可以为所有的张量静态分配内存，避免动态内存分配带来的开销。

add.cpp的实现代码如下所示：

```
#include "./add.h"          //IR注册头文件
#include <vector>         //可使用vector类模板并调用vector相关接口。
#include <string>         //C++标准库，可使用string类构造对象并调用string相关接口

namespace ge {
bool InferShapeAndTypeAdd(Operator& op,
    const string& inputName1,
    const string& inputName2, const string& outputName)
 {
    TensorDesc vOutputDesc = op.GetOutputDescByName(outputName.c_str());
```

```
    DataType inputDtype = op.GetInputDescByName(inputName1.c_str()).GetDataType();
    Format inputFormat = op.GetInputDescByName(inputName1.c_str()).GetFormat();
    // 针对shape维度大小进行交换
    ge::Shape shapeX = op.GetInputDescByName(inputName1.c_str()).GetShape();
    ge::Shape shapeY = op.GetInputDescByName(inputName2.c_str()).GetShape();
    std::vector<int64_t> dimsX = shapeX.GetDims();
    std::vector<int64_t> dimsY = shapeY.GetDims();
    if (dimsX.size() < dimsY.size()) {
        std::vector<int64_t> dimsTmp = dimsX;
        dimsX = dimsY;
        dimsY = dimsTmp;
    }

   // 对小的shape进行1补齐
   if (dimsX.size() != dimsY.size()) {
    int dec = dimsX.size() - dimsY.size();
    for (int i = 0; i < dec; i++) {
      dimsY.insert(dimsY.begin(), (int64_t)1);
    }
   }

    // 设置输出的shape维度
    std::vector<int64_t> dimVec;
    for (size_t i = 0; i < dimsX.size(); i++) {
        if ((dimsX[i] != dimsY[i]) && (dimsX[i] != 1) && (dimsY[i] != 1)) {
            return false;
        }

        int64_t dims = dimsX[i] > dimsY[i] ? dimsX[i] : dimsY[i];
        dimVec.push_back(dims);
    }
    ge::Shape outputShape = ge::Shape(dimVec);

    vOutputDesc.SetShape(outputShape);
    vOutputDesc.SetDataType(inputDtype);
    vOutputDesc.SetFormat(inputFormat);
    op.UpdateOutputDesc(outputName.c_str(), vOutputDesc);

    return true;
}

//----------------Add-------------------
IMPLEMT_VERIFIER(Add, AddVerify)
{
    if (op.GetInputDescByName("x1").GetDataType() != op.GetInputDescByName("x2").GetDataType()) {
        return GRAPH_FAILED;
    }
    return GRAPH_SUCCESS;
}


// Obtains the processing function of the output tensor description.
IMPLEMT_COMMON_INFERFUNC(AddInferShape)
{
    if(InferShapeAndTypeAdd(op, "x1", "x2", "y")) {
        return GRAPH_SUCCESS;
    }
    return GRAPH_FAILED;
}


//Registered inferfunction，第一个参数为算子的OpType
COMMON_INFER_FUNC_REG(Add, AddInferShape);


//Registered verify function，第一个参数为算子的OpType
VERIFY_FUNC_REG(Add, AddVerify);
//----------------Add-------------------
}
```

8.6.3 算子信息定义

需要通过配置算子信息文件，将算子的相关信息注册到算子信息库中。算子信息库主要体现算子在昇腾AI处理器上物理实现的限制，包括算子的输入输出dtype、format以及输入shape信息。网络运行时，FE会根据算子信息库中的算子信息做基本校验，判断是否需要为算子插入合适的转换节点，并根据算子信息库中信息找到对应的算子实现文件进行编译，生成算子二进制文件进行执行。

进入“tbe/op_info_cfg/ai_core/*soc_version*”目录，配置算子信息库文件“add.ini”。

*soc_version*为当前昇腾AI处理器的版本，例如ascend310、ascend910等。

开发者需要基于MindStudio自动生成的add.ini文件进行修改，修改后的Add算子的算子信息定义如下所示。

```
[Add]
input0.name=x1
input0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
input0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
input0.shape=all
input0.paramType=required
input1.name=x2
input1.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
input1.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
input1.shape=all
input1.paramType=required
output0.name=y
output0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32
output0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND
output0.shape=all
output0.paramType=required
opFile.value=add
opInterface.value=add
```

参数说明请参见**表8-14**。下表仅列出常用的算子信息库配置项，其他配置项请参见

以下手册中的算子信息库定义章节。

- 《**TBE自定义算子开发指南(推理)**》
- 《**TBE自定义算子开发指南(训练)**》

表 8-14 Add 算子的算子信息定义说明

信息	Add算子配置	说明
[OpType]	[Add]	算子类型，以英文半角方括号，标识一个算子信息开始，根据**8.3 算子分析**，算子类型为Add。
input0.name	x1	Add算子的第一个输入tensor的名称，根据**8.3 算子分析**，第一个输入名称为x1，所以此处配置为x1，且需要跟**算子原型定义**中的名称保持一致。
input0.dtype	input0.dtype=float16,float16,float16,float16,float,float,flo	定义输入tensor支持的数据类型与数据排布格式。

input0.format	at,float,int32,int32,int32,int32 input0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND	根据8.3 算子分析，Add算子的输入数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW、NC1HWC0、NHWC、ND。 注意： 若算子输入支持多种规格，算子输入的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
input0.shape	all	定义输入tensor支持的形状。
input0.paramType	required	定义输入tensor的类型。 ● dynamic：表示该输入是动态个数，可能是1个，也可能是多个。 ● optional：表示该输入为可选，可以有1个，也可以不存在。 ● required：表示该输入有且仅有1个。 Add算子的input0为固定输入1个，此处配置为required。
input1.name	x2	因为Add算子有两个输入，生成的配置模板中只有一个输入，所以此处需要添加input1的相关配置。 此配置项代表Add算子的第二个输入tensor的名称，根据8.3 算子分析，第二个输入名称为x2，所以此处配置为x2，且需要跟算子原型定义中的名称保持一致。
input1.dtype input1.format	input1.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32 input1.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND	定义输入tensor支持的数据类型与数据排布格式。 根据8.3 算子分析，Add算子的输入数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW,NC1HWC0,NHWC,ND。 注意： 若算子输入支持多种规格，算子输入的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
input1.shape	all	定义输入tensor支持的形状。

版权所有 © 华为技术有限公司

input1.paramType	required	定义输入tensor的类型。 ● dynamic：表示该输入是动态个数，可能是1个，也可能是多个。 ● optional：表示该输入为可选，可以有1个，也可以不存在。 ● required：表示该输入有且仅有1个。 Add算子的input1为固定输入1个，此处配置为required。
output0.name	y	Add算子的输出tensor的名称，根据**8.3 算子分析**，算子的输出名称为y，所以此处配置为y，且需要跟**算子原型定义**中的名称保持一致。
output0.dtype output0.format	output0.dtype=float16,float16,float16,float16,float,float,float,float,int32,int32,int32,int32 output0.format=NCHW,NC1HWC0,NHWC,ND,NCHW,NC1HWC0,NHWC,ND	定义输出tensor支持的数据类型与数据排布格式 根据**8.3 算子分析**，Add算子的输出数据类型支持float16、float32与int32三种；支持的数据排布格式有NCHW,NC1HWC0,NHWC,ND。 注意： 若算子输出支持多种规格，算子输出的dtype与format需要一一对应、按对应顺序进行配置，列出算子支持的所有dtype与format的组合，中间以“,”分隔。
output0.shape	all	定义输出tensor支持的形状。
output0.paramType	required	定义输出tensor的类型。 ● dynamic：表示该输出是动态个数，可能是1个，也可能是多个。 ● optional：表示该输出为可选，可以有1个，也可以不存在。 ● required：表示该输出有且仅有1个。 Add算子的output0为固定输出1个，此处配置为required。
opFile.value	add	算子代码实现文件的名称。 因为算子实现代码的名称符合算子OpType到代码实现名称的转换规则，参见**步骤4**，所以此字段可不配置。

opInterface.value	add	算子实现文件中定义函数的名称。 因为算子定义函数的名称符合算子OpType到算子定义函数名称的转换规则，参见步骤4，所以此字段可不配置。

8.6.4 UT 测试

8.6.4.1 简介

MindStudio提供了基于gtest框架的新的UT测试方案，简化了开发者开发UT测试用例的复杂度。

UT（Unit Test：单元测试）是开发人员进行单算子运行验证的手段之一，主要目的是：

- 测试算子代码的正确性，验证输入输出结果与设计的一致性。
- UT侧重于保证算子程序能够跑通，选取的场景组合应能覆盖算子代码的所有分支（一般来说覆盖率要达到100%），从而降低不同场景下算子代码的编译失败率。

说明

测试类的详细定义可参见***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*|*{arch}*-*linux*/toolkit/python/site-packages/op_test_frame/ut/op_ut.py**文件。

8.6.4.2 接口参考

8.6.4.2.1 OpUT

概述

OpUT为UT测试框架的基类，提供了测试用例定义及测试用例执行的接口，主要包含如下两个接口：

- add_case
- add_precision_case

OpUT 测试类定义

- 函数原型

  OpUT(op_type, op_module_name=None, op_func_name=None)
- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add（文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见步骤4，例如BiasAdd算子自动生成module名称"impl.bias_add"。

- op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

```
@check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
```

add_case 接口

- 函数原型

  OpUT.add_case(support_soc=None, case=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - case：该参数为dict类型，示例如下：

```
{
  "params": [
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    },
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    }
  ],
  "case_name": "test_add_case_1",
  "expect": "success"
}
```

    该dict中key字段含义如下：

    - params：该字段在测试用例运行时透传给算子接口。
    - case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

      test_[op_type]_auto_case_name_[case_count]

      例如： test_Add_auto_case_name_1
    - expect：期望结果。默认为期望“success”，也可以是预期抛出的异常，例如RuntimeError。

add_precision_case 接口

- 函数原型

  OpUT.add_precision_case(support_soc=None, case)

- 功能说明

  添加算子编译+精度测试的用例。

- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - case：该参数为dict类型，示例如下：

```
{
   "params": [
      {
        "shape": (32, 64),
        "ori_shape": (32, 64),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16",
        "param_type": "input"
      },
      {
        "shape": (32, 64),
        "ori_shape": (32, 64),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16",
        "param_type": "output"
      }
   ],
   "case_name": "test_add_case_1",
   "calc_expect_func": np_add #一个函数
   "precision_standard": precision_info.PrecisionStandard(0.001, 0.001)  #可选字段
}
```

    该dict中key字段含义如下：

    - params：该字段在测试用例运行时传给算子接口。
    - case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

      test_[op_type]_auto_case_name_[case_count]

      例如： test_Add_auto_case_name_1
    - calc_expect_func：期望结果生成函数。
    - precision_standard：自定义精度标准，若不配置此字段，按照如下默认精度与期望数据进行比对：
      - float16：双千分之一，即每个数据之间的误差不超过千分之一，误差超过千分之一的数据总和不超过总数据数的千分之一。
      - float32：双万分之一，即每个数据之间的误差不超过万分之一，误差超过万分之一的数据总和不超过总数据数的万分之一。

- 调用示例

  示例1：

```
from op_test_frame.ut import OpUT
# "ut_case" 为UT测试框架的关键字，不可修改
ut_case = OpUT("Add", "impl.add", "add")

def np_add(x1, x2, y):
    y = (x1.get("value") + x2.get("value"), )
```

```
    return y

ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add
})

if __name__ == '__main__':
    ut_case.run("Ascend910",None,"ca","/home/allan/Ascend/toolkit/tools/simulator")
```

说明：

ut_case.run：请参见**run接口**。

此样例未设置算子的输入数据值，测试框架将默认使用np.random.uniform(value_range, size=shape).astype(dtype)为每一个输入自动生成输入数据。

其中value_range使用默认值[0.1, 1.0]，shape和dtype的取值为“params”中的每一个输入。value_range也可以通过参数指定，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value_range": [2.0, 3.0]
}
```

也可以指定输入的值，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value": np.zeros((32, 64), np.float16)
}
```

示例2：

自定义精度标准示例如下：

```
from op_test_frame.common import precision_info
from op_test_frame.ut import OpUT

ut_case = OpUT("Add")
ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add,
    "precision_standard": precision_info.PrecisionStandard(0.1, 0.1) # 使用精度标准双十分之一，与期望
数据进行比对
})
```

run 接口

- 函数原型

  OpUT.run(soc, case_name=None, simulator_mode=None, simulator_lib_path=None)

- 功能说明

  执行测试用例。

- 参数说明
  - soc：执行测试用例的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-*linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。
  - case_name：指定执行的case，配置为**add_case接口**及**add_precision_case接口**中的“case_name”
  - simulator_lib_path: 仿真库所在的路径。

    该路径结构如下所示：

    ```
    simulator_lib_path/
      Ascend910/
        lib/
          libpv_model.so
          ...
      Ascend310/
        lib/
    ```

```
libpv_model.so
...
```

须知

使用MindStudio运行UT测试用例时，无需用户手工调用OpUT.run接口。

8.6.4.2.2 BroadcastOpUT

概述

BroadcastOpUT继承了OpUT，包含了OpUT的能力。

BroadcastOpUT主要供双输入单输出的Broadcast类型的算子进行测试用例的定义，例如Add、Mul等算子。BroadcastOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

BroadcastOpUT 测试类定义

- 函数原型

  BroadcastOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如BiasAdd算子自动生成module名称"impl.bias_add"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_broadcast_case 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case(self, soc, input_1_info, input_2_info, output_info=None,expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - input_1_info：算子的第一个输入的信息，有两种形式：

    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - input_2_info: 算子的第二个输入的信息，与input_1_info含义相同。
  - output_info：默认为None，不需要填写。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND"], ["float16", (32, 32), "ND"])

ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND", (32, 32), "ND"], ["float16", (32, 32),
"ND", (32, 32), "ND"])

# 期望异常的用例
ut_case.add_broadcast_case("all", ["float16", (31, 32), "ND"], ["float16", (32, 32), "ND"],
expect=RuntimeError)
```

add_broadcast_case_simple 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case_simple(self, soc, dtypes, shape1, shape2, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。此接口较add_broadcast_case更加简化。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}-linux***/**atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape1: 算子的第一个输入的shape。
  - shape2：算子的第二个输入的shape。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_broadcast_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_broadcast_case_simple(["Ascend910", "Ascend310"], ["float16", "float32"], (32, 32), (32,
32))
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
```

```
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.6.4.2.3 ElementwiseOpUT

概述

ElementwiseOpUT继承了OpUT，包含了OpUT的能力。

ElementwiseOpUT主要供单输入单输出的Elementwise类型的算子进行测试用例的定义，例如Abs, Square等算子。

ElementwiseOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ElementwiseOpUT 测试类定义

- 函数原型

  ElementwiseOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如MaximumGrad算子自动生成module名称"impl.maximum_grad"。

- op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

```
@check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
```

add_elewise_case 接口

- 函数原型

  BroadcastOpUT.add_elewise_case(self, soc, param_info, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - param_info：算子的输入信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_elewise_case("Ascend910", ["float16", (32, 32), "ND"])
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})
```

add_elewise_case_simple 接口

- 函数原型

  BroadcastOpUT.add_elewise_case_simple(self, soc, dtypes, shape, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape：算子输入的shape。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_elewise_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_elewise_case_simple("Ascend910", ["float16", "float32"], [32, 32])
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.6.4.2.4 ReduceOpUT

概述

ReduceOpUT继承了OpUT，包含了OpUT的能力。

ReduceOpUT主要供Reduce类型的算子进行测试用例的定义，例如ReduceSum, ReduceMean等算子。ReduceOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ReduceOpUT 测试类定义

- 函数原型

  ReduceOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add（文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如ReduceSum算子自动生成module名称"impl.reduce_sum"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_reduce_case 接口

- 函数原型

  BroadcastOpUT.add_reduce_case(self, soc, input_info, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - input_info：算子的输入信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - axes: reduce的轴信息。
  - keep_dims: 是否保持reduce的轴为1。True：是；False：否。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

  ```
  ut_case.add_reduce_case("Ascend910", ["float16", (32, 32), "ND"], [0,], False)
  ```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32,),
        "ori_shape": (32,),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, [0,], False]
})
```

add_reduce_case_simple 接口

- 函数原型

  BroadcastOpUT.add_reduce_case_simple(self, soc, dtypes, shape, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape：算子输入的shape。
  - axes: reduce的轴信息。
  - keep_dims: 是否保持reduce的轴为1。True：是；False：否。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_reduce_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_reduce_case_simple("Ascend910", ["float16", "float32"], [32, 32], [1,], True)
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 1),
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
```

```
    }, [1,], True]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 1),
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, [1,], True]
})
```

8.6.4.3 创建和运行 UT 测试用例

前提条件

需完成自定义算子的开发，包括算子实现代码、算子原型定义，详情可参见**8.5.1 算子代码实现**、**8.5.2 算子原型定义**、**8.5.3 算子信息定义**。

操作步骤

步骤1 创建UT测试用例。

1. 创建UT测试用例，有以下三种入口：

   右键单击算子工程根目录，选择“**New Cases > UT Case**”。

   若已经存在了算子的UT测试用例，可以右键单击“**testcases**”目录，或者“**testcases > ut**”目录，选择“**New Cases > UT Case**”，创建UT测试用例。

2. 在弹出的算子选择界面，选择需要创建UT测试用例的算子，单击**OK**，如下图所示。

说明

若已存在此算子的UT测试用例，系统会提示“testcases/ut/ops_test/xx already exists. Do you want to overwrite?”

您可以选择“Overwrite”或者“Cancel”。

创建完成后，会在算子工程根目录下生成testcases文件夹，目录结构如下所示：

```
├── MyOperator          //工程根目录
│   ├── testcases
│   │   ├── libs            // gtest框架，为第三方依赖，用户无需关注
│   │   ├── ut
│   │   │   ├── ops_test
│   │   │   │   ├── add
│   │   │   │   │   ├── CMakeLists.txt      //用于编译可执行文件
```

```
│   │   │   │   │   ├── test_add_impl.py      //算子实现代码的测试用例文件
│   │   │   │   │   ├── test_add_proto.cpp    //算子原型定义代码的测试用例文件
│   │   │   │   ├── CMakeLists.txt            //用于编译可执行文件
│   │   │   │   ├── test_main.cpp             //测试用例调用总入口
│   │   │   ├ CMakeLists.txt
```

步骤2 编写算子实现代码的UT Python测试用例。

在“testcases/ut/ops_test/add/test_add_impl.py”文件中，编写算子实现代码的UT Python测试用例，计算出算子执行结果，并取回结果和预期结果进行比较，来测试算子逻辑的正确性。

```
import sys
from op_test_frame.ut import BroadcastOpUT  # 导入UT测试类，可根据算子类型选择使用哪个测试类。

ut_case = BroadcastOpUT("Add")      # 实例化UT测试用例，ut_case为UT测试框架关键字，不可修改；Add为算子的Type

def calc_expect_func(input_x, input_y, output_z):    # 自定义实现生成期望数据的函数
    res = input_x["value"] + input_y["value"]
    return [res, ]    # 返回期望数据

# 添加测试用例
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "output"}],
    "calc_expect_func": calc_expect_func
})
# 若定义多个用例，定义多个ut_case.add_precision_case函数
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "input"},
               {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
                "param_type": "output"}],
    "calc_expect_func": calc_expect_func
})
```

1. 首先导入UT测试类，用户可根据算子类型自行选择使用哪个UT测试类，详细可参见**8.5.5.2 接口参考**。
2. 实例化测试用例，OpUT的使用方法可参见**OpUT测试类定义**。
3. 用户自定义实现生成期望数据的函数。
4. 添加测试用例。

   可参见**8.5.5.2 接口参考**查看每个测试类接口的使用方法。

   若要与期望数据进行结果的比对，请使用**add_precision_case接口**。

步骤3 编写算子原型定义的UT C++测试用例。

在“testcases/ut/ops_test/add/test_add_proto.cpp”文件中，编写算子原型定义的UT C++测试用例，用于定义算子实例、更新算子输入输出并调用InferShapeAndType函数，最后验证InferShapeAndType函数执行过程及结果的正确性。

1. 导入gtest测试框架和算子IR定义的头文件。

   UT的C++用例采用的是gtest框架，所以需要导入gtest测试框架；算子原型定义在原型定义头文件中，所以需要导入原型定义的*.h文件。

```
//导入gtest框架
#include <gtest/gtest.h>
//导入基础的vector类库
```

```
#include <vector>
//导入算子的IR定义头文件
#include "add.h"
```

2. 定义测试类。

   UT的C++用例采用的是gtest框架，所以需要定义一个类来继承gtest的测试类。

```
#include <gtest/gtest.h>
#include <vector>
#include "add.h"

class AddTest : public testing::Test {
protected:
    static void SetUpTestCase() {
        std::cout << "add test SetUp" << std::endl;
}

    static void TearDownTestCase() {
        std::cout << "add test TearDown" << std::endl;
    }
};
```

   测试类的名称可自定义，以“Test”为后缀。

3. 编写测试用例。

   每一个场景写一个测试用例函数，该用例中需要构造算子实例，包括算子名称、shape、数据类型。然后调用InferShapeAndType函数，并将推导出的shape、dtype与预期结果进行对比。

   示例如下：

```
TEST_F(AddTest, add_test_case_1) {
    // 定义算子实例及输入shape和type，以TensorDesc实例承载
    ge::op::Add add_op;  //Add为算子的Type，需要与原型定义的REG_OP(OpType)中的OpType保持一致。
    ge::TensorDesc tensorDesc;
    ge::Shape shape({2, 3, 4});
    tensorDesc.SetDataType(ge::DT_FLOAT16);
    tensorDesc.SetShape(shape);

    // 更新算子输入，输入的名称需要与原型定义*.h文件中的名称保持一致，例如:x1与x2分别为Add算子的两个输入
    add_op.UpdateInputDesc("x1", tensorDesc);
    add_op.UpdateInputDesc("x2", tensorDesc);
    // 调用InferShapeAndType函数，InferShapeAndType()接口为固定接口，用例执行时会自动调用算子原型定义中的shape推导函数
    auto ret = add_op.InferShapeAndType();
    // 验证调用过程是否成功
    EXPECT_EQ(ret, ge::GRAPH_SUCCESS);

    // 获取算子输出并比较shape和type，算子输出的名字需要与原型定义*.h文件中的名称保持一致，例如:算子的输出为y
    auto output_desc = add_op.GetOutputDesc("y");
    EXPECT_EQ(output_desc.GetDataType(), ge::DT_FLOAT16);
    std::vector<int64_t> expected_output_shape = {2, 3, 4};
    EXPECT_EQ(output_desc.GetShape().GetDims(), expected_output_shape);
}
```

   若不同输入的shape不同，请自行定义多个TensorDesc对象进行设置，例如：

```
    ge::op::Operator1 operator1_op;  //Operator1为算子的Type
    ge::TensorDesc tensorDesc1;
    ge::TensorDesc tensorDesc2;
    ge::Shape shape1({2, 3, 4});
    ge::Shape shape2({3, 4, 5});
    tensorDesc1.SetDataType(ge::DT_FLOAT16);
    tensorDesc1.SetShape(shape1);
    tensorDesc2.SetDataType(ge::DT_FLOAT16);
    tensorDesc2.SetShape(shape2);

    // 更新算子输入
    operator1_op.UpdateInputDesc("x1", tensorDesc1);
    operator1_op.UpdateInputDesc("x2", tensorDesc2);
```

步骤4 运行算子实现文件的UT测试用例。

开发人员可以执行当前工程中所有算子的UT测试用例，也可以执行单个算子的UT测试用例。

- 右键单击“testcases/ut/ops_test”文件夹，选择**Run Tbe Operator‘All’UT Impl with coverage**，执行整个文件夹下算子实现代码的测试用例。
- 右键单击“testcases/ut/ops_test/*算子名称*”文件夹，选择**Run Tbe Operator‘*算子名称*’UT Impl with coverage**，执行单个算子实现代码的测试用例。

  第一次运行时会弹出运行配置页面，请参考配置，然后单击**Run**。后续如需修改运行配置，请参考13.3.1 修改运行配置。

  表 8-15 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_impl。
SoC Version	下拉选择当前版本的昇腾AI处理器类型。
Target	运行环境。 – Simulator_Function：功能仿真环境。 – Simulator_TMModel：快速展示算子执行的调度流水线，不进行实际算子计算。 **当前功能仿支持Ascend 310和Ascend 910系列AI处理器。**
Operator Name	选择运行的测试用例。 – all表示运行所有用例。 – 其他表示运行某个算子下的测试用例。
Case Names	勾选需要运行的测试用例，即算子实现代码的UT Python测试用例。支持全选和全不选所有测试用例。

- 查看运行结果。
  a. 运行完成后，通过界面下方的“Run”日志打印窗口查看运行结果。
  b. 在“Run”窗口中单击index.html的URL（URL中的localhost为MindStudio安装服务器的IP，建议直接单击打开），查看UT测试用例的覆盖率结果，如图8-20所示。

  说明

  查看UT测试用例运行结果需要使用浏览器，如果未安装浏览器，请用户自行安装。

图 8-20 查看 UT 用例覆盖率结果

Coverage report: 100%

Module ↑	statements	missing	excluded	coverage
/home/[redacted]/MindStudio/samples/demo_projects/custom/tbe/impl/add.py	43	0	0	100%
Total	**43**	**0**	**0**	**100%**

表示算子的UT用例覆盖率为100%，UT验证通过。

如果出现"Page'http://***.html'requested without authorization, you can copy URL and open it in browser to trust it."提示，请参考**15.8.3 配置不受信任的网址访问浏览器**解决。

c. 在html页面中单击对应算子，进入UT用例覆盖率详情页面，如**图8-21**所示，通过绿色和红色标签区分是否覆盖。

图 8-21 UT 覆盖率详情页面

Coverage for **/home/[redacted]/MindStudio/samples/demo_projects/custom/tbe/impl/add.py** : 100%

43 statements | 43 run | 0 missing | 0 excluded

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  """
4  Copyright (C) 2019. Huawei Technologies Co., Ltd. All rights reserved.
5
6  This program is free software; you can redistribute it and/or modify
7  it under the terms of the Apache License Version 2.0.You may not use this file
8  except in compliance with the License.
9
10 This program is distributed in the hope that it will be useful,
11 but WITHOUT ANY WARRANTY; without even the implied warranty of
12 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 Apache License for more details at
14 http://www.apache.org/licenses/LICENSE-2.0
15
16 add
17 """
18 from __future__ import absolute_import
19
20 import te.lang.cce
21 from te import tvm
22 from te.platform.fusion_manager import fusion_manager
23 from topi import generic
24 from topi.cce import util
```

d. 运行完成后，生成结果如下所示：

- 如果在配置运行信息选择的**Target**为**Simulator_TMModel**，还可以查看执行流水线，如下图所示。

说明

在芯片进行运算前，vector、cube、MTE1、MTE2、MTE3等单元会做初始化操作，对应在timeline显示上会出现数据还未搬运，各个单元就产生流水数据。

步骤5 运行算子原型定义的UT测试用例。

开发人员可以执行当前工程中所有算子的UT测试用例，也可以执行单个算子的UT测试用例。

- 右键单击“testcases/ut/ops_test”文件夹，选择**Run Tbe Operator‘All’UT Proto**，执行整个文件夹下算子原型定义代码的测试用例。
- 右键单击“testcases/ut/ops_test/*算子名称*”文件夹，选择**Run Tbe Operator‘*算子名称*’UT Proto**，执行单个算子原型定义代码的测试用例。

  第一次运行时会弹出运行配置页面，请参考配置，然后单击**Run**。后续如需修改运行配置，请参考**13.3.1 修改运行配置**。

  表 8-16 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_proto。
Operator Name	选择运行的测试用例。 – all表示运行所有用例。 – 其他表示运行某个算子下的测试用例。
Case Names	勾选需要运行的测试用例，即TEST_F中定义的用例。支持全选和全不选所有测试用例。

运行完成后，通过界面下方的日志打印窗口，查看运行结果。结果中展示此次一共运行几个用例，成功几个，失败几个，如下图所示。

----结束

8.6.5 算子工程编译

简介

将自定义算子工程编译生成自定义算子安装包custom_opp_*Target OS*_*Target Architecture*.run。

具体编译内容为：将算子插件实现文件、算子原型定义文件、算子信息定义文件分别编译成算子插件、算子原型库、算子信息库。编译过程示意如图8-22。

图 8-22 编译过程

编译操作

步骤1 在MindStudio工程界面，选中算子工程。

步骤2 单击顶部菜单栏的“ Build > Edit Build Configuration...”。

步骤3 进入编译配置页面，如下图所示。请参考表8-17进行编译配置。

图 8-23 编译配置界面

表 8-17 编译配置参数说明

参数	说明
Build Configuration	编译配置名称，默认为Build-Configuration

参数	说明
Build Mode	编译方式。 ● Remote Build：远端编译。 **须知** 远端编译需要g++版本为7.5.0。 ● Local Build：本地编译。 算子工程在MindStudio安装服务器进行编译，方便用户通过编译日志快速定位到MindStudio的实现代码所在位置，从而快速定位问题。此种方式下需要配置交叉编译环境，编译环境的配置请参考**2.3.5 配置编译环境**。
SSH Connection	Remote Build模式下显示该配置。下拉选择SSH配置信息，若未添加配置信息，请单击 ⊞ 添加。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。
Target OS	Local Build模式下显示该配置。 ● 针对Ascend EP：选择昇腾AI处理器所在硬件环境的Host侧的操作系统。 ● 针对Ascend RC：选择板端环境的操作系统。
Target Architecture	Local Build模式下显示该配置。选择Target OS的操作系统架构。

步骤4 单击“Build”进行工程编译。

步骤5 在界面最下方的窗口查看编译结果，并在算子工程的cmake-build目录下生成自定义算子安装包**custom_opp_*Target OS*_*Target Architecture*.run**。

其中**Target OS** 与**Target Architecture**的取值参见**步骤3**中的配置。

----结束

8.6.6 算子部署（本地）

步骤1 在MindStudio工程界面菜单栏依次选择“Ascend > Deploy”。

弹出算子部署界面。

步骤2 在弹出的界面中选择“Deploy Locally”，并单击“Deploy”按钮。

在下方Output页签出现如下信息，代表自定义算子部署成功。

```
Output:  Output   Detail
2021-01-07 14:31:45  Uncompressing version:1.0  100%
2021-01-07 14:31:45  [runtime]  [INFO] copy uninstall sh success
2021-01-07 14:31:45  [ops_custom]upgrade framework
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops framework files ......
2021-01-07 14:31:45  [runtime]  copy new ops framework files ......
2021-01-07 14:31:45  [ops_custom]upgrade op proto
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops op_proto files ......
2021-01-07 14:31:45  [runtime]  copy new ops op_proto files ......
2021-01-07 14:31:45  [ops_custom]upgrade op impl
2021-01-07 14:31:45  [runtime]  [INFO] replace old ops op_impl files ......
2021-01-07 14:31:45  [runtime]  copy new ops op_impl files ......
2021-01-07 14:31:45  [ops_custom]changemode...
2021-01-07 14:31:45  SUCCESS
Output  Log  6: TODO  4: Run  Profiling  Terminal  Pylint  Build
```

自定义算子包安装成功后，会将自定义算子部署在***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux/opp**目录下的对应文件夹中。

目录结构示例如下所示：

```
├── opp      //算子库目录
│   ├── op_impl
│     ├── built-in
│     ├── custom
│       ├── ai_core
│         ├── tbe
│           ├── config
│             ├── soc_version     //昇腾AI处理器版本
│               ├── aic-soc_version-ops-info.json     //自定义算子信息库文件
│           ├── custom_impl            //自定义算子实现代码文件
│             ├── add.py
│       ├── vector_core   //此目录预留，无需关注
│   ├── framework
│     ├── built-in
│     ├── custom
│       ├── caffe
│       ├── tensorflow          //存放tensorflow框架的自定义算子插件库
│         ├── libcust_tf_parsers.so
│         ├── npu_supported_ops.json   //Ascend 910场景下使用的文件，Ascend 310场景下无需关注
│   ├── op_proto
│     ├── built-in
│     ├── custom
│       ├── libcust_op_proto.so    //自定义算子原型库文件
```

----结束

8.6.7 算子部署（远程）

将自定义算子安装包**custom_opp_*Targert OS*_*Target Architecture*.run**部署到昇腾AI处理器所在硬件环境的算子库中，为后续算子在网络中运行构造必要条件。

步骤1 在MindStudio工程界面，选中算子工程。

步骤2 单击顶部菜单栏的“Ascend > Deploy”，进入算子打包部署界面。

请在**Deploy Remotely > SSH Connection**中下拉选择SSH配置信息，若未添加SSH配置信息，请单击 ⊞ 添加。

步骤3 配置环境变量

有两种方式：

- 在硬件设备的Host侧配置环境变量。

  MindStudio使用Host侧的运行用户在Host侧进行算子部署，进行算子部署执行前，需要在Host侧进行如下环境变量的配置。

  a. 以运行用户在Host侧的$HOME/.bashrc文件中配置如下环境变量。

  ```
  export ASCEND_OPP_PATH=/home/xxx/Ascend/opp
  ```

  /home/xxx/Ascend/为OPP组件（算子库）的安装路径，请根据实际情况配置。

  b. 执行命令使环境变量生效。

  source ~/.bashrc

- 在**Environment Variables**中添加环境变量。

  可以在**Environment Variables**中直接输入ASCEND_OPP_PATH=*/home/xxx/Ascend/opp*。

  */home/xxx/Ascend/*为OPP组件（算子库）的安装路径，请根据实际情况配置。

  也可以点击文本框后的图标，在弹出的对话框中填写。

  – 在Name中输入环境变量名称：ASCEND_OPP_PATH。
  – 在Value中输入环境变量值：*/home/xxx/Ascend/opp*。

图 8-24 算子打包部署界面

步骤4 选择算子部署的目标服务器，单击“Deploy”。

步骤5 算子部署过程即**8.5.6 算子工程编译**生成的自定义算子安装包的安装过程，部署完成后，算子被部署在Host侧算子库OPP对应文件夹中，默认路径为/usr/local/Ascend/opp/。

图 8-25 算子部署日志打印

```
2020-03-09 21:49:59  Verifying archive integrity... All good.
2020-03-09 21:49:59  Uncompressing version:1.0
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] copy uninstall sh success
2020-03-09 21:49:59  [ops_custom]upgrade framework
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops framework files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops framework files ......
2020-03-09 21:49:59  [ops_custom]upgrade op proto
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops op_proto files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops op_proto files ......
2020-03-09 21:49:59  [ops_custom]upgrade op impl
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] [INFO] replace old ops op_impl files ......
2020-03-09 21:49:59  [runtime] [2020-03-10 09:49:57] copy new ops op_impl files ......
2020-03-09 21:49:59  [ops_custom]changemode...
2020-03-09 21:49:59  SUCCESS
2020-03-09 21:49:59
2020-03-09 21:49:59  2020-03-10 09:49:59 - [INFO] Deploying the operator info to the remote host finished.
```

Host侧自定义算子部署完成后目录结构示例如下所示：

```
├── opp      //算子库目录
│   ├── op_impl
```

```
|       ├── built-in
|       ├── custom
|          ├── ai_core
|             ├── tbe
|                ├── config
|                   ├── aic_ops_info.json     //自定义算子信息库文件
|                ├── custom_impl              //自定义算子实现代码文件
|                   ├── add.py
|          ├── vector_core   //此目录预留，无需关注
|    ├── framework
|       ├── built-in
|       ├── custom
|          ├── caffe
|          ├── tensorflow          //存放tensorflow框架的自定义算子插件库
|             ├── libcust_tf_parsers.so
|    ├── op_proto
|       ├── built-in
|       ├── custom
|          ├── libcust_op_proto.so    //自定义算子原型库文件
```

----结束

8.6.8 PyTorch 适配

说明

MindStudio暂不支持PyTorch算子适配，在完成TBE算子开发后，Pytorch适配请参见《**PyTorch算子开发指南**》。

PyTorch适配请用户根据《**PyTorch算子开发指南**》完成以下操作。

- 安装PyTorch依赖环境
- 编译和安装PyTorch框架
- PyTorch框架下适配TBE算子

8.6.9 ST 测试

概述

MindStudio提供了新的ST（System Test）测试框架，可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性和计算结果准确性，并生成运行测试报告，包括：

- 基于算子信息库生成算子测试用例定义文件。
- 基于算子测试用例定义文件生成不同shape、dtype的测试数据和基于AscendCL的测试用例。
- 编译算子工程并将算子部署到算子库，最后在硬件环境中执行测试用例，验证算子运行的正确性。
- 自动生成运行报表（st_report.json）功能，报表记录了测试用例信息及各阶段运行情况。
- 根据用户定义并配置的算子期望数据生成函数，回显期望算子输出和实际算子输出的对比测试结果，验证计算结果的准确性。

前提条件

- 完成自定义算子的开发，请参见**8.6.1 算子代码实现**、**8.6.2 算子原型定义**、**8.6.3 算子信息定义**、**8.6.8 PyTorch适配**。

- MindStudio已连接硬件设备。

生成 ST 测试用例定义文件

步骤1 创建ST测试用例。

有以下三种入口：

- 右键单击算子工程根目录，选择“**New Cases > ST Case**”。
- 右键单击算子信息定义文件：***{工程名}*** **/tbe /op_info_cfg/ai_core/*{SoC Version}* /*xx*.ini**，选择“**New Cases > ST Case**”。
- 若已经存在了对应算子的ST Case，可以右键单击“**testcases**”目录，或者“**testcases > st**”目录，选择“**New Cases > ST Case**”，追加ST测试用例。

步骤2 在弹出的“Create ST Cases for an Operator”界面中选择需要创建ST测试用例的算子。

如下图所示：

Operator：下拉选择算子名称。

SoC Version：下拉选择昇腾AI处理器的类型。

- 若不勾选“Import operator info from a model”，单击“OK”后，会生成shape为空的算子测试用例定义文件，Design视图如下所示：

用户需要进行shape信息的配置，用于生成测试数据及测试用例，您也可以根据需要进行其他字段的配置，每个字段的详细要求可参见**算子测试用例定义文件参数解释**。

- 若用户勾选“Import operator info from a model”，选择包含算子的onnx模型文件（*.onnx）后，界面会显示获取到的模型文件的首层shape信息。

  用户也可以在“Input Nodes Shape”中修改首层输入的shape信息。

  单击“OK”后，工具会自动根据首层shape信息dump出选择算子的shape信息，生成对应的算子测试用例定义文件，测试用例定义文件有Text和Design两种视图，Design视图如下所示：

版权所有 © 华为技术有限公司

此文件用于生成测试数据及测试用例，您可以参见**算子测试用例定义文件参数解释**进行相关字段的修改。

步骤3 若要将算子与标杆数据对比，需要定义并配置算子期望数据生成函数。

1. 自定义实现算子期望数据生成函数。

   算子期望数据生成函数是用TensorFlow或Caffe等框架实现的与自定义算子功能相同的函数，其可以在CPU上运行并生成标杆数据。标杆数据用来与自定义算子生成数据进行对比，根据对比结果确定自定义算子精度。算子期望数据生成函数用python语言实现，在一个python文件中可以实现多个算子期望数据生成函数。该函数的输入、输出、属性与自定义算子的输入、输出、属性的Format、Type、Shape保持一致。

2. 配置算子测试用例定义文件。

   在算子测试用例文件中配置算子期望数据生成函数。有两种场景，分别为Design视图和Text视图下进行配置。

版权所有 © 华为技术有限公司

– 若在Design视图下，配置方法为在Expected Result Verification下的Script Path中选择算子期望数据生成函数的python文件所在路径。在Script Function中输入算子期望数据生成函数的函数名。

说明

Script Function文本框中可以输入算子期望数据生成函数的函数名，也可以为空。

- 当输入算子期望数据生成函数的函数名时，ST测试时会使用该函数名对应的函数生成标杆数据。
- 当输入为空时，会自动匹配Script Path路径下的python文件中与自定义算子同名的函数，用于ST测试时生成标杆数据。若无同名函数，则会提示匹配失败。

– 若在Text视图下，增加"calc_expect_func_file"参数，其值为算子期望数据生成函数对应的文件路径及算子函数名，如：

```
"calc_expect_func_file": "/home/teste/test_*.py:function",  //配置期望算子文件
```

其中，/home/teste/test_*.py为算子期望数据生成函数的实现文件，function为对应的函数名称。算子期望数据生成函数文件路径和函数名称用冒号隔开。

步骤4 修改Case信息后，单击“Save”，修改会保存到算子测试用例定义文件。

算子测试用例定义文件存储目录为算子工程根目录下的“testcases/st/*OpType*/*{SoC Version}*”文件夹下，命名为*OpType*_case_*timestamp*.json

说明

请勿更改算子测试用例定义文件命名格式和将其他文件的以该命名格式保存在算子工程根目录下的“testcases/st/*OpType*/*{SoC Version}*”文件夹下。否则会导致文件解析错误。

----结束

运行 ST 测试用例

步骤1 右键单击**生成ST测试用例定义文件**中生成的ST测试用例定义文件（路径：“testcases > st > add > *{SoC Version}* > xxxx.json”），选择 "Run The Operator 'xxx' ST Case"。

表 8-18 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择st_cases 。
Execute Mode	• Remote Execute 远程执行测试 • Local Execute 本地执行测试

参数	说明
SSH Connection	当Execute Mode选择Remote Execute时，下拉选择SSH配置信息，若未添加配置信息，请单击 ⊞ 添加。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。
Environment Variables	● 在文本框中添加环境变量 *PATH_1=路径1；PATH_2=路径2* 多个环境变量用英文分号隔开。 ● 也可以点击文本框后的图标，在弹出的对话框中填写。 – 在Name中输入环境变量名称：PATH_1。 – 在Value中输入环境变量值：*路径1*。 勾选Instead system environment variables可以显示系统环境变量。
Test Case Id	选择需要运行的算子。
SoC Version	选择处理器版本。
Executable File Name	下拉选择需要执行的测试用例定义文件。
Target OS	针对Ascend EP：选择昇腾AI处理器所在硬件环境的Host侧的操作系统。 针对Ascend RC：选择板端环境的操作系统。
Target Architecture	选择Target OS的操作系统架构。
Case Names	选择运行的Case Name。 **说明**：默认全选所有用例，可以去除勾选部分不需要运行的用例。

步骤2 Host侧运行用户设置。

Host侧运行用户需在SSH Connection中添加，并且该用户必须为HwHiAiUser属组。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。

步骤3 在运行环境上有两种方式配置相关组件的环境变量。

- 在远程设备上配置环境变量。
  - 针对Ascend EP，需要在硬件设备的Host侧配置安装组件路径的环境变量。

    以Host侧运行用户在~/.bashrc文件中配置acllib或fwkacllib、driver组件的安装路径。

    i. 打开运行用户下的.bashrc文件。

    vi ~/.bashrc

    ii. 在文件最后添加如下信息。

    export ASCEND_DRIVER_PATH=*/usr/local/Ascend/driver*

    export ASCEND_HOME=*/usr/local/Ascend*/ascend-toolkit/latest

    export ASCEND_AICPU_PATH=${ASCEND_HOME}/*<target architecture>*

若远程设备为推理环境：

export LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64:${ASCEND_HOME}/acllib/lib64:$LD_LIBRARY_PATH

若远程设备为训练环境：

export LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64/driver:${ASCEND_DRIVER_PATH}/lib64/common:${ASCEND_HOME}/fwkacllib/lib64:$LD_LIBRARY_PATH

若已存在如上所述环境变量，请确认为当前运行环境实际安装组件所在路径。

<target architecture>为操作系统架构，例如：x86_64-linux或arm64-linux。

iii. 保存退出。

:wq

iv. 使环境变量生效。

source ~/.bashrc

- 在本地配置环境变量

  在**步骤1**中运行的窗口中Environment Variables参数进行环境变量配置。

  – 在文本框中添加环境变量

  ```
  ASCEND_DRIVER_PATH=/usr/local/Ascend/driver;
  ASCEND_HOME=/usr/local/Ascend/ascend-toolkit/latest;
  ASCEND_AICPU_PATH=${ASCEND_HOME}/<target architecture>;
  若远程设备为推理环境：
  LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64:${ASCEND_HOME}/acllib/lib64:$LD_LIBRARY_PATH;
  若远程设备为训练环境：
  LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64/driver:${ASCEND_DRIVER_PATH}/lib64/common:${ASCEND_HOME}/fwkacllib/lib64:$LD_LIBRARY_PATH;
  ```

  若已存在如上所述环境变量，请确认为当前运行环境实际安装组件所在路径。

  – 也可以点击文本框后的图标，在弹出的对话框中填写。

  分别在Name中输入环境变量名称，在Value中输入环境变量值。

步骤4 单击“**Run**”。

MindStudio会根据算子测试用例定义文件在“*算子工程根目录*/testcases/st/out/*OpType*”下生成测试数据和测试代码，并编译出可执行文件，在指定的硬件设备上执行测试用例，并将执行结果和与标杆数据对比报告打印到Output窗口中，同时在“*算子工程根目录*/testcases/st/out/*OpType*”下生成st_report.json文件。st_report.json文件信息含义请参见**表8-19**。

表 8-19 st_report.json 报表主要字段及含义

字段				说明
run_cmd	-	-	-	命令行命令。
report_list	-	-	-	报告列表，该列表中可包含多个测试用例的报告。

字段				说明
	trace_detail	-	-	运行细节。
		st_case_info	-	测试信息。
			expect_data_path	期望计算结果路径。
			case_name	测试用例名称。
			input_data_path	输入数据路径。
			planned_output_data_paths	实际计算结果输出路径。
			op_params	算子参数信息。
		stage_result	-	运行各阶段结果信息。
	case_name	-	-	测试名称。
	status	-	-	测试结果信息。

----结束

其他功能扩展

- 支持Design和Text界面视图的切换

  双击“testcases/st/*OpType*/*{SoC Version}*”文件夹下的算子测试用例定义文件或者右键该文件选择"Design Testcase *OpType*_case_*timestamp*.json"，会打开下图界面，该界面中的Design和Text均为json文件的一个视图。

  在Design视图中的更改，若该视图不存在错误配置信息，切换到Text视图时，更改会自动同步到Text视图中；同理，在Text视图中的更改，切换到Design视图时，更改也会同步到Design视图中，如下图所示：

- 新增测试用例

  在算子测试用例定义界面单击“Add”按钮，弹出如下图所示弹框，提示用户输入“Case Name”。

"Case Name"为由数字、字母、下划线组成的字符串。

版权所有 © 华为技术有限公司

单击“OK”后，会在Design Cases页面的尾部新增创建好的Case，新增Case的各个字段值为空，需要用户进行配置，配置规则可参见**算子测试用例定义文件参数解释**。

- 删除测试用例

  在算子测试用例定义界面，单击相应Case右侧的图标，即可删除该Case。

- 选择单个或多个测试用例运行

  在算子测试用例定义界面，勾选其中一个或多个Case，单击“Run”，即可运行选中的Case。

算子测试用例定义文件参数解释

表 8-20 算子测试用例定义 json 文件

参数		说明
Test_xxx_xx	-	必选。string类型。 测试用例的名称。

参数		说明
Input[xx]	-	必选。 算子的输入。 **须知** 一个算子所有Input中参数取值的个数都要一致，否则测试用例生成会失败。 例如：Input[01]的format支持的类型个数2，则Input[02]的format支持的类型个数也需要为2。 同理，所有Input[xx]中的type、shape、data_distribute和value_range的取值个数也需要保持一致。
-	Format	必选。 String或者String的一维数组。 输入tensor数据的排布格式，不允许为空。 支持如下数据排布格式： ● NCHW ● NHWC ● ND：表示支持任意格式。 ● NC1HWC0：华为自研的5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。 ● FRACTAL_Z：卷积的权重的格式。 ● FRACTAL_NZ：华为自研的分形格式，在cube单元计算时，输出矩阵的数据格式为NW1H1H0W0。整个矩阵被分为（H1*W1）个分形，按照column major排布，形状如N字形；每个分形内部有（H0*W0）个元素，按照row major排布，形状如z字形。考虑到数据排布格式，将NW1H1H0W0数据格式称为Nz格式。其中，H0,W0表示一个分形的大小，示意图如下所示：

参数		说明
-	Type	必选。 String或者String的一维数组。 输入数据支持的数据类型。 ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16 ● float
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状。
-	ValueRange	必选。 int类型或者float类型，一维或者二维数组。 取值范围，不能为空。 为[min_value, max_value]且min_value <=max_value。

参数		说明
-	DataDistribute	必选。 String或者String的一维数组。 使用哪种数据分布方式生成测试数据，支持的分布方式有： ● uniform：返回均匀分布随机值 ● normal：返回正态分布（高斯分布）随机值 ● beta：返回Beta分布随机值 ● laplace：返回拉普拉斯分布随机值 ● triangular：返回三角形分布随机值 ● relu：返回均匀分布+Relu激活后的随机值 ● sigmoid：返回均匀分布 + sigmoid激活后的随机值 ● softmax：返回均匀分布 + softmax激活后的随机值 ● tanh：返回均匀分布 + tanh激活后的随机值
Output[xx]	-	必选。 算子输出。 **须知** Output中参数取值的个数都要与Input一致，否则测试用例生成会失败。 例如：Input的format支持的类型个数2，则Output的format支持的类型个数也需要为2。

参数		说明
-	Format	必选。 String或者String的一维数组。 输出tensor数据的排布格式，不允许为空。 支持如下数据排布格式： ● NCHW ● NHWC ● ND：表示支持任意格式。 ● NC1HWC0：华为自研的5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。 ● FRACTAL_Z：卷积的权重的格式。 ● FRACTAL_NZ：华为自研的分形格式，在cube单元计算时，输出矩阵的数据格式为NW1H1H0W0。整个矩阵被分为（H1*W1）个分形，按照column major排布，形状如N字形；每个分形内部有（H0*W0）个元素，按照row major排布，形状如z字形。考虑到数据排布格式，将NW1H1H0W0数据格式称为Nz格式。其中，H0,W0表示一个分形的大小，示意图如下所示：

参数		说明
-	Type	必选。 String或者String的一维数组。 输出数据支持的数据类型： ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16 ● float
-	Shape	必选。 int类型，一维或者二维数组。 输出tensor支持的形状。
Attr	-	可选。
-	Name	若配置attr，则为必选。 string类型。 属性的名称，不为空。
-	Type	若配置attr，则为必选属性支持的类型。 string类型。 输出数据支持的数据类型： ● bool ● int ● float ● string ● list_bool ● list_int ● list_float ● list_string ● list_list_int

参数		说明
-	Value	若配置attr，则为必选，且不允许为null。 string类型。 属性值，根据type的不同，属性值不同。 ● bool: true/false ● int: 10 ● float: 1.0 ● string: “NCHW” ● list_bool: [false, true] ● list_int: [1, 224, 224, 3] ● list_float: [1.0, 0.0] ● list_string: [“str1”, “str2”] ● list_list_int: [[1, 3, 5, 7], [2, 4, 6, 8]]
Expected Result Verification	-	可选。
-	Script Path	算子期望数据生成函数路径。
-	Script Function	算子期望数据生成函数名称。

8.7 TBE 算子开发（MindSpore）

8.7.1 算子实现和注册信息文件

通过调用TBE DSL接口，在算子工程下的“mindspore/impl/cus_square_impl.py”文件中完成Square算子实现和注册信息文件代码编写，包括实现TBE算子、注册算子。具体实现方式请参考**MindSpore教程>自定义算子（Ascend）**中的“实现TBE算子和注册算子信息”小节。

说明

MindSporeTBE算子开发需要运行环境上安装MindSpore框架，安装方法请参见**https://www.mindspore.cn/install/**。

8.7.2 算子原型定义

在算子工程下的“op_proto/cus_square.py”文件中完成算子原语文件编写，将算子注册到算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。具体实现方式请参考**MindSpore教程>自定义算子（Ascend）**中的“注册算子原语”小节。

8.7.3 UT 测试

8.7.3.1 简介

MindStudio提供了基于gtest框架的新的UT测试方案，简化了开发者开发UT测试用例的复杂度。

UT（Unit Test：单元测试）是开发人员进行单算子运行验证的手段之一，主要目的是：

- 测试算子代码的正确性，验证输入输出结果与设计的一致性。
- UT侧重于保证算子程序能够跑通，选取的场景组合应能覆盖算子代码的所有分支（一般来说覆盖率要达到100%），从而降低不同场景下算子代码的编译失败率。

说明

测试类的详细定义可参见***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}*-*linux*/toolkit/python/site-packages/op_test_frame/ut/op_ut.py**文件。

8.7.3.2 接口参考

8.7.3.2.1 OpUT

概述

OpUT为UT测试框架的基类，提供了测试用例定义及测试用例执行的接口，主要包含如下两个接口：

- add_case
- add_precision_case

OpUT 测试类定义

- 函数原型

  OpUT(op_type, op_module_name=None, op_func_name=None)
- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如BiasAdd算子自动生成module名称"impl.bias_add"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_case 接口

- 函数原型

OpUT.add_case(support_soc=None, case=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - case：该参数为dict类型，示例如下：

```
{
  "params": [
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    },
    {
      "shape": (32, 64),
      "ori_shape": (32, 64),
      "format": "ND",
      "ori_format": (32, 64),
      "dtype": "float16"
    }
  ],
  "case_name": "test_add_case_1",
  "expect": "success"
}
```

    该dict中key字段含义如下：

    - params：该字段在测试用例运行时透传给算子接口。
    - case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

      test_[op_type]_auto_case_name_[case_count]

      例如： test_Add_auto_case_name_1
    - expect：期望结果。默认为期望“success”，也可以是预期抛出的异常，例如RuntimeError。

add_precision_case 接口

- 函数原型

  OpUT.add_precision_case(support_soc=None, case)

- 功能说明

  添加算子编译+精度测试的用例。

- 参数说明
  - support_soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。

– case：该参数为dict类型，示例如下：

```
{
    "params": [
        {
          "shape": (32, 64),
          "ori_shape": (32, 64),
          "format": "ND",
          "ori_format": "ND",
          "dtype": "float16",
          "param_type": "input"
        },
        {
          "shape": (32, 64),
          "ori_shape": (32, 64),
          "format": "ND",
          "ori_format": "ND",
          "dtype": "float16",
          "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add #一个函数
    "precision_standard": precision_info.PrecisionStandard(0.001, 0.001)  #可选字段
}
```

该dict中key字段含义如下：

- params：该字段在测试用例运行时传给算子接口。
- case_name：测试用例的名称，可选参数。若不设置，测试框架会自动生成用例名称，生成规则如下：

  test_[op_type]_auto_case_name_[case_count]

  例如： test_Add_auto_case_name_1
- calc_expect_func：期望结果生成函数。
- precision_standard：自定义精度标准，若不配置此字段，按照如下默认精度与期望数据进行比对：
  - float16：双千分之一，即每个数据之间的误差不超过千分之一，误差超过千分之一的数据总和不超过总数据数的千分之一。
  - float32：双万分之一，即每个数据之间的误差不超过万分之一，误差超过万分之一的数据总和不超过总数据数的万分之一。

- 调用示例

  示例1：

```
from op_test_frame.ut import OpUT
# "ut_case" 为UT测试框架的关键字，不可修改
ut_case = OpUT("Add", "impl.add", "add")

def np_add(x1, x2, y):
    y = (x1.get("value") + x2.get("value"), )
    return y

ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
```

```
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add
})

if __name__ == '__main__':
    ut_case.run("Ascend910",None,"ca","/home/allan/Ascend/toolkit/tools/simulator")
```

说明：

ut_case.run：请参见**run接口**。

此样例未设置算子的输入数据值，测试框架将默认使用np.random.uniform(value_range, size=shape).astype(dtype)为每一个输入自动生成输入数据。

其中value_range使用默认值[0.1, 1.0]，shape和dtype的取值为“params”中的每一个输入。value_range也可以通过参数指定，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value_range": [2.0, 3.0]
}
```

也可以指定输入的值，如下所示：

```
{
    "shape": (32, 64),
    "ori_shape": (32, 64),
    "format": "ND",
    "ori_format": "ND",
    "dtype": "float16",
    "param_type": "input",
    "value": np.zeros((32, 64), np.float16)
}
```

示例2：

自定义精度标准示例如下：

```
from op_test_frame.common import precision_info
from op_test_frame.ut import OpUT

ut_case = OpUT("Add")
ut_case.add_precision_case(case={
    "params": [
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
```

```
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "input"
        },
        {
            "shape": (32, 64),
            "ori_shape": (32, 64),
            "format": "ND",
            "ori_format": "ND",
            "dtype": "float16",
            "param_type": "output"
        }
    ],
    "case_name": "test_add_case_1",
    "calc_expect_func": np_add,
    "precision_standard": precision_info.PrecisionStandard(0.1, 0.1) # 使用精度标准双十分之一，与期望
数据进行比对
})
```

run 接口

- 函数原型

  OpUT.run(soc, case_name=None, simulator_mode=None, simulator_lib_path=None)

- 功能说明

  执行测试用例。

- 参数说明
  - soc：执行测试用例的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-*linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。
  - case_name：指定执行的case，配置为**add_case接口**及**add_precision_case接口**中的“case_name”
  - simulator_lib_path: 仿真库所在的路径。

    该路径结构如下所示：

    ```
    simulator_lib_path/
      Ascend910/
        lib/
            libpv_model.so
            ...
      Ascend310/
        lib/
            libpv_model.so
            ...
    ```

须知

使用MindStudio运行UT测试用例时，无需用户手工调用OpUT.run接口。

8.7.3.2.2 BroadcastOpUT

概述

BroadcastOpUT继承了OpUT，包含了OpUT的能力。

BroadcastOpUT主要供双输入单输出的Broadcast类型的算子进行测试用例的定义，例如Add、Mul等算子。BroadcastOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

BroadcastOpUT 测试类定义

- 函数原型

  BroadcastOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如BiasAdd算子自动生成module名称"impl.bias_add"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_broadcast_case 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case(self, soc, input_1_info, input_2_info, output_info=None,expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从ATC安装路径的“atc/data/platform_config”目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - input_1_info：算子的第一个输入的信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - input_2_info: 算子的第二个输入的信息，与input_1_info含义相同。
  - output_info：默认为None，不需要填写。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。

– case_name: 默认为None，测试框架会自动生成测试用例的名称。

示例：

```
ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND"], ["float16", (32, 32), "ND"])

ut_case.add_broadcast_case("all", ["float16", (32, 32), "ND", (32, 32), "ND"], ["float16", (32, 32),
"ND", (32, 32), "ND"])

# 期望异常的用例
ut_case.add_broadcast_case("all", ["float16", (31, 32), "ND"], ["float16", (32, 32), "ND"],
expect=RuntimeError)
```

add_broadcast_case_simple 接口

- 函数原型

  BroadcastOpUT.add_broadcast_case_simple(self, soc, dtypes, shape1, shape2, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。此接口较add_broadcast_case更加简化。

- 参数说明

  – soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。

  – dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。

  – shape1: 算子的第一个输入的shape。

  – shape2：算子的第二个输入的shape。

  – expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。

  – case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_broadcast_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_broadcast_case_simple(["Ascend910", "Ascend310"], ["float16", "float32"], (32, 32), (32,
32))
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
```

```
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc=["Ascend910", "Ascend310"], case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.7.3.2.3 ElementwiseOpUT

概述

ElementwiseOpUT继承了OpUT，包含了OpUT的能力。

ElementwiseOpUT主要供单输入单输出的Elementwise类型的算子进行测试用例的定义，例如Abs, Square等算子。ElementwiseOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ElementwiseOpUT 测试类定义

- 函数原型

  ElementwiseOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明
  - op_type: 算子的类型。
  - op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如MaximumGrad算子自动生成module名称"impl.maximum_grad"。
  - op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

    ```
    @check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
    def add(input_x, input_y, output_z, kernel_name="add"):
    ```

add_elewise_case 接口

- 函数原型

BroadcastOpUT.add_elewise_case(self, soc, param_info, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}*-*linux***/**atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - param_info：算子的输入信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_elewise_case("Ascend910", ["float16", (32, 32), "ND"])
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})
```

add_elewise_case_simple 接口

- 函数原型

  BroadcastOpUT.add_elewise_case_simple(self, soc, dtypes, shape, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录***/**ascend-toolkit/*{version}*/*{arch}*-*linux***/**atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。

- dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
- shape：算子输入的shape。
- expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
- case_name: 默认为None，测试框架会自动生成测试用例的名称。

此接口较add_elewise_case接口，相当于将输入的所有format配置为“ND”。

示例：

```
ut_case.add_elewise_case_simple("Ascend910", ["float16", "float32"], [32, 32])
```

以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }]
})
```

8.7.3.2.4 ReduceOpUT

概述

ReduceOpUT继承了OpUT，包含了OpUT的能力。

ReduceOpUT主要供Reduce类型的算子进行测试用例的定义，例如ReduceSum, ReduceMean等算子。ReduceOpUT为这类算子提供了更加便利的接口，例如，创建算子编译用例时，对于一些简单场景无需输入format等信息。

ReduceOpUT 测试类定义

- 函数原型

  ReduceOpUT(op_type, op_module_name=None, op_func_name=None)

- 参数说明

- op_type: 算子的类型。
- op_module_name: 算子的module名称（即算子的实现文件名称和路径），例如：impl.add （文件路径为：impl/add.py）。默认值为None，可根据op_type自动生成，op_type到算子实现文件名称的转换规则可参见**步骤4**，例如ReduceSum算子自动生成module名称"impl.reduce_sum"。
- op_func_name: 算子的接口名称，算子实现文件中的算子接口名。默认值为None，可根据op_type自动生成，op_type到算子实现文件中接口名称的转换规则可参见**步骤4**，例如Add算子，则需要在 impl/add.py 中存在add接口，接口定义如下所示：

```
@check_op_params(REQUIRED_INPUT, REQUIRED_INPUT, REQUIRED_OUTPUT, KERNEL_NAME)
def add(input_x, input_y, output_z, kernel_name="add"):
```

add_reduce_case 接口

- 函数原型

  BroadcastOpUT.add_reduce_case(self, soc, input_info, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-*linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - input_info：算子的输入信息，有两种形式：
    - [dtype, shape, format, ori_shape, ori_format]
    - [dtype, shape, format]：此种形式下，ori_shape与ori_format的取值与shape、format的取值相同。
  - axes: reduce的轴信息。
  - keep_dims: 是否保持reduce的轴为1。True：是；False：否。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  示例：

```
ut_case.add_reduce_case("Ascend910", ["float16", (32, 32), "ND"], [0,], False)
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32,),
        "ori_shape": (32,),
        "format": "ND",
        "ori_format": "ND",
```

```
        "dtype": "float16"
    }, [0,], False]
})
```

add_reduce_case_simple 接口

- 函数原型

  BroadcastOpUT.add_reduce_case_simple(self, soc, dtypes, shape, axes, keep_dim=False, expect=op_status.SUCCESS, case_name=None)

- 功能说明

  添加算子编译的测试用例，测试算子是否支持相关规格，编译出“.o”文件。

- 参数说明
  - soc：测试该用例是否支持对应的昇腾AI处理器，昇腾AI处理器的取值范围可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/atc/data/platform_config**目录下查看，对应“*.ini”文件的名字即为可配置的昇腾AI处理器类型。support_soc支持的数据类型为str、tuple或者list，tuple或者list表示可以支持多个SoC。若配置为“all”或者“None”，表示支持所有的SoC。
  - dtypes：需要测试的数据类型，填写多个数据，相当于一次添加了多个测试用例。
  - shape：算子输入的shape。
  - axes: reduce的轴信息。
  - keep_dims: 是否保持reduce的轴为1。True：是；False：否。
  - expect： 期望是否编译成功，默认为“op_status.SUCCESS”，当测试异常场景时，期望的结果可配置为RuntimeError。
  - case_name: 默认为None，测试框架会自动生成测试用例的名称。

  此接口较add_reduce_case接口，相当于将输入的所有format配置为“ND”。

  示例：

```
ut_case.add_reduce_case_simple("Ascend910", ["float16", "float32"], [32, 32], [1,], True)
```

  以上用例与调用如下add_case接口的用例实现功能相同：

```
ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, {
        "shape": (32, 1),
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float16"
    }, [1,], True]
})

ut_case.add_case(support_soc="Ascend910", case={
    "params": [{
        "shape": (32, 32),
        "ori_shape": (32, 32),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, {
        "shape": (32, 1),
```

```
        "ori_shape": (32, 1),
        "format": "ND",
        "ori_format": "ND",
        "dtype": "float32"
    }, [1,], True]
})
```

8.7.3.3 创建和运行 UT 测试用例

前提条件

MindSpore框架的算子测试，需完成MindSpore算子工程中相应的交付件，详情可参见**8.7.1 算子实现和注册信息文件**、**8.7.2 算子原型定义**。

操作步骤

步骤1 创建UT测试用例。

1. 创建UT测试用例，有以下三种入口：

   右键单击算子工程根目录，选择“**New Cases > UT Case**”。

   若已经存在了算子的UT测试用例，可以右键单击“**testcases**”目录，或者“**testcases > ut**”目录，选择“**New Cases > UT Case**”，创建UT测试用例。

2. 在弹出的算子选择界面，选择需要创建UT测试用例的算子*cus_square*，单击**OK**。

   说明

   若已存在此算子的UT测试用例，系统会提示“testcases/ut/ops_test/xx already exists. Do you want to overwrite?”

   您可以选择“Overwrite”或者“Cancel”。

   MindSpore框架的算子测试，创建完成后，会在算子工程根目录下生成testcases文件夹，目录结构如下所示：

```
├── MyOperator          //工程根目录
│   ├── testcases
│       ├── ut
│           ├── ops_test
│               ├── test_cus_square_impl.py      //算子实现代码的测试用例文件
```

步骤2 编写算子实现代码的UT Python测试用例。

在“testcases/ut/ops_test/test_*cus_square*_impl.py”文件中，编写算子实现代码的UT Python测试用例，计算出算子执行结果，并取回结果和预期结果进行比较，来测试算子逻辑的正确性。

```
import sys
import numpy as np
from op_test_frame.ut import BroadcastOpUT    # 导入UT测试类，可根据算子类型选择使用哪个测试类。

# 实例化UT测试用例，ut_case为UT测试框架关键字，不可修改；
# 修改op_func_name的值，与cus_square_impl.py中注册算子函数名相同。
ut_case = BroadcastOpUT("cus_square_impl", op_func_name="CusSquareImpl")

# 自定义实现生成期望数据的函数
def calc_expect_func(input_x, output_y):
    res = np.square(input_x["value"])
    return [res, ]


# 添加测试用例
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
                "param_type": "input"},
```

```
            {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (32,), "shape": (32,),
             "param_type": "output"}],
    "calc_expect_func": calc_expect_func
})

# 若定义多个用例，定义多个ut_case.add_precision_case函数
ut_case.add_precision_case("all", {
    "params": [{"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
             "param_type": "input"},
            {"dtype": "float16", "format": "ND", "ori_format": "ND", "ori_shape": (16,2), "shape": (16,2),
             "param_type": "output"},],
    "calc_expect_func": calc_expect_func
})
```

1. 首先导入UT测试类，用户可根据算子类型自行选择使用哪个UT测试类，详细可参见**8.7.3.2 接口参考**。
2. 实例化测试用例，OpUT的使用方法可参见**OpUT测试类定义**。
3. 用户自定义实现生成期望数据的函数。
4. 添加测试用例。

   可参见**8.7.3.2 接口参考**查看每个测试类接口的使用方法。

   若要与期望数据进行结果的比对，请使用**add_precision_case接口**。

步骤3 运行算子实现文件的UT测试用例。

开发人员可以执行当前工程中所有算子的UT测试用例，也可以执行单个算子的UT测试用例。

- 右键单击“testcases/ut/ops_test”文件夹，选择**Run Tbe Operator‘All’UT Impl with coverage**，执行整个文件夹下算子实现代码的测试用例。
- 右键单击“testcases/ut/ops_test/test_*cus_square*_impl.py”文件夹，选择**Run Tbe Operator‘*算子名称*’UT Impl with coverage**，执行单个算子实现代码的测试用例。

  第一次运行时会弹出运行配置页面，请参考配置，然后单击**Run**。后续如需修改运行配置，请参考**13.3.1 修改运行配置**。

表 8-21 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_impl。
SoC Version	下拉选择当前版本的昇腾AI处理器类型。 MindSpore只支持昇腾910处理器。
Target	运行环境。 – Simulator_Function：功能仿真环境。 – Simulator_TMModel：快速展示算子执行的调度流水线，不进行实际算子计算。 **当前功能仅支持Ascend 910系列AI处理器。**
Operator Name	选择运行的测试用例。 – all表示运行所有用例。 – 其他表示运行某个算子下的测试用例。

参数	说明
Case Names	勾选需要运行的测试用例，即算子实现代码的UT Python测试用例。支持全选和全不选所有测试用例。

- 查看运行结果。
  a. 运行完成后，通过界面下方的“Run”日志打印窗口查看运行结果。
  b. 在“Run”窗口中单击index.html的URL（URL中的localhost为MindStudio安装服务器的IP，建议直接单击打开），查看UT测试用例的覆盖率结果，如图8-26所示。

  说明

  查看UT测试用例运行结果需要使用浏览器，如果未安装浏览器，请用户自行安装。

  图 8-26 查看 UT 用例覆盖率结果

  Coverage report: 100%

  filter...

Module ↑	statements	missing	excluded	coverage
/home/[redacted]/mindspore_opreator_sample/mindspore/impl/cus_square_impl.py	21	0	0	100%
Total	**21**	**0**	**0**	**100%**

  表示算子的UT用例覆盖率为100%，UT验证通过。

  如果出现图8-27提示，请参考15.8.3 配置不受信任的网址访问浏览器解决。

  图 8-27 提示

  Page 'http://localhost:63343/t...rage_report/ut/index.html' requested without authorization, you can copy URL and open it in browser to trust it.

  Copy authorization URL to clipboard | No

  c. 在html页面中单击对应算子，进入UT用例覆盖率详情页面，如图8-28所示，通过绿色和红色标签区分是否覆盖。

  图 8-28 UT 覆盖率详情页面

  Coverage for **/home/[redacted]/mindspore_opreator_sample/mindspore/impl/cus_square_impl.py** : 100%

  21 statements | 21 run | 0 missing | 0 excluded

```
1 from __future__ import absolute_import
2 from te import tvm
3 from topi import generic
4 import te.lang.cce
5 from topi.cce import util
6 from mindspore.ops.op_info_register import op_info_register, TBERegOp, DataType
```

  d. 运行完成后，生成的中间文件与可执行文件目录结构如下所示：

```
├── MyOperator          //工程根目录
│   ├── out
│   │   ├── bin
│   │   │   ├── data
│   │   │   ├── kernel_meta
│   │   │   ├── report
│   │   │   ├── task_scheduler.dump
│   │   ├── coverage_report
│   │   │   ├── ut_impl
│   │   ├── model
```

```
│   │   ├──── **.json
│   │   ├──── **.json
```

- 如果在配置运行信息选择的**Target**为**Simulator_TMModel**，还可以查看执行流水线，如下图所示。

说明

在芯片进行运算前，vector、cube、MTE1、MTE2、MTE3等单元会做初始化操作，对应在timeline显示上会出现数据还未搬运，各个单元就产生流水数据。

----结束

8.7.4 ST 测试

概述

MindStudio提供了MindSpore框架的算子ST（System Test）测试框架，可以自动生成测试用例，在真实的硬件环境中，验证算子功能的正确性和计算结果准确性，包括：

- 根据算子实现和算子信息文件（*_impl.py）生成算子测试用例定义文件（*.json），作为算子ST测试用例的输入。
- 根据算子测试用例定义文件生成ST测试数据及测试用例执行代码，在硬件环境上执行算子测试用例。

前提条件

- 完成自定义算子的开发，详情可参见**8.7.1 算子实现和注册信息文件**、**8.7.2 算子原型定义**。
- 服务器上已经安装MindSpore环境，并且有昇腾910系列处理器。

生成 ST 测试用例定义文件

步骤1 创建ST测试用例。

有以下三种入口：

- 右键单击算子工程根目录，选择“**New Cases > ST Case**”。
- 右键单击算子实现和算子信息文件：***{工程名}*/mindspore/impl/*cus_square*_impl.py**，选择“**New Cases > ST Case**”。
- 若已经存在了对应算子的ST Case，可以右键单击“**testcases**”目录，或者“**testcases > st**”目录，选择“**New Cases > ST Case**”，追加ST测试用例。

步骤2 在弹出的“Create ST Cases for an Operator”界面中选择需要创建ST测试用例的算子。

如下图所示：

Operator：下拉选择算子名称。

单击“OK”后，会生成shape为空的算子测试用例定义文件，Design视图如下所示：

用户需要进行shape信息的配置，用于生成测试数据及测试用例，您也可以根据需要进行其他字段的配置。单击 Add 可以添加测试用例。每个字段的详细要求可参见**表8-23**。

步骤3 修改Case信息后，单击“Save”，修改会保存到算子测试用例定义文件。

算子测试用例定义文件存储目录为算子工程根目录下的“testcases/st/*OpType*”文件夹下，命名为*OpType*_case_*timestamp*.json。

说明

请勿更改算子测试用例定义文件命名格式和将其他文件的以该命名格式保存在算子工程根目录下的“testcases/st/*OpType*”文件夹下。否则会导致文件解析错误。

----结束

运行 ST 测试用例

步骤1 右键单击ST测试用例定义文件（路径：“testcases > st > cus_square> xxxx.json”），选择 "Run The Operator 'xxx' ST Case"。

表 8-22 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择st_cases 。
Execute Mode	● Remote Execute 远程执行测试 ● Local Execute 本地执行测试
SSH Connection	当Execute Mode选择Remote Execute时，下拉选择SSH配置信息，若未添加配置信息，请单击 ⊞ 添加。添加SSH配置信息的方法请参见**13.1 SSH连接管理**。
Test Case Id	选择需要运行的算子。
Executable File Name	下拉选择需要执行的测试用例定义文件。
Target OS	针对Ascend EP：选择昇腾AI处理器所在硬件环境的Host侧的操作系统。 针对Ascend RC：选择板端环境的操作系统。 目前只支持EP模式。
Target Architecture	选择Target OS的操作系统架构。
Case Names	选择运行的Case Name。 **说明**：默认全选所有用例，可以去除勾选部分不需要运行的用例。

步骤2 单击“**Run**”。

MindStudio会根据算子测试用例定义文件在“*算子工程根目录*/testcases/st/out/*OpType*”下生成测试数据和测试代码，在指定的硬件设备上执行测试用例，并将执行结果打印到Output窗口中。

----结束

表 8-23 MindSpore 算子测试用例定义 json 文件

参数		说明	类型	是否必选
case_name		测试用例的名称。	String类型	是

参数		说明	类型	是否必选
st_mode		ST测试模式，其值为："ms_python_train"，表示Mindspore的算子工程。	String类型	是
op		算子的类型。 不允许为空。	String类型	是
input_desc		算子输入描述。 **须知** 所有input_desc中参数取值的个数都要一致，否则测试用例生成会失败。 例如：input1的format支持的类型个数2，则intput2的format支持的类型个数也需要为2。 同理，所有input*x*中的type、shape、data_distribute和value_range的取值个数也需要保持一致。		是
	type	输入数据支持的数据类型。 ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16/fp16/half ● float32/float ● double ● UNDEFINED：预留，当type配置为该值，则format必须配置为“RESERVED”，代表算子的此输入可选。	String或者String的一维数组	是
	shape	输入tensor支持的形状。	int类型。 一维或者二维数组。	是

参数		说明	类型	是否必选
	data_distribute	使用哪种数据分布方式生成测试数据，支持的分布方式有： ● uniform：返回均匀分布随机值 ● normal：返回正态分布（高斯分布）随机值 ● beta：返回Beta分布随机值 ● laplace：返回拉普拉斯分布随机值 ● triangular：返回三角形分布随机值 ● relu：返回均匀分布+Relu激活后的随机值 ● sigmoid：返回均匀分布 + sigmoid激活后的随机值 ● softmax：返回均匀分布 + softmax激活后的随机值 ● tanh：返回均匀分布 + tanh激活后的随机值	String或者String的一维数组	是
	value_range	取值范围，不能为空。 为[min_value, max_value]且min_value <=max_value。	int类型或者float类型。 一维或者二维数组。	是
output_desc		算子输出描述。 **须知** output_desc中参数取值的个数都要与input_desc一致，否则测试用例生成会失败。 例如：input*x*的format支持的类型个数2，则output的format支持的类型个数也需要为2。		是

参数		说明	类型	是否必选
	type	输出数据支持的数据类型。 ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16/fp16/half ● float32/float ● double ● UNDEFINED：预留，当type配置为该值，则format必须配置为“RESERVED”，代表算子的此输入可选。	String或者String的一维数组	是
	shape	输出tensor支持的形状。	int类型。 一维或者二维数组。	是
attr				否
	name	属性的名称，不为空。	String	若配置attr，则为必选

参数		说明	类型	是否必选
	type	属性支持的类型。 ● bool ● int ● float ● string ● list_bool ● list_int ● list_float ● list_string ● list_list_int ● data_type：数据类型	String	若配置attr，则为必选
	value	属性值，根据type的不同，属性值不同。 ● bool: true/false ● int: 10 ● float: 1.0 ● string: “NCHW” ● list_bool: [false, true] ● list_int: [1, 224, 224, 3] ● list_float: [1.0, 0.0] ● list_string: [“str1”, “str2”] ● list_list_int: [[1, 3, 5, 7], [2, 4, 6, 8]] ● data_type：DT_FLOAT data_type支持的取值范围可参见ATC所在路径下“include/graph/types.h”文件中的“DataType”类型的定义。	String	若配置attr，则为必选，且不允许为null。

8.8 AI CPU 算子开发（TensorFlow）

8.8.1 算子代码实现

简介

AI CPU算子的实现包括两部分：

- 头文件（.h文件）：进行算子类的声明，自定义算子类需要继承CpuKernel基类。
- 源文件（.cc文件）：重写算子类中的Compute函数，进行算子计算逻辑的实现。

头文件代码模块介绍

用户需要在算子工程的“cpukernel/impl/reshape_cust_kernel.h”文件中进行算子类的声明，代码模块介绍如下所示：

```
#ifndef _RESHAPE_CUST_KERNELS_H_
#define _RESHAPE_CUST_KERNELS_H_

#include "cpu_kernel.h"        // CpuKernel基类以及注册宏定义
// 以下头文件根据算子需求引入
#include "cpu_tensor.h"        // Tensor定义以及相关方法
#include "cpu_tensor_shape.h"  // Tensor shape的定义以及相关方法
#include "cpu_types.h"         // 数据类型以及格式等定义
#include "cpu_attr_value.h"    // AttrValue定义及相关方法

namespace aicpu {        // 定义命名空间aicpu
class ReshapeCustCpuKernel : public CpuKernel {        // ReshapeCust算子类继承CpuKernel基类
public:
    ~ReshapeCustCpuKernel() = default;
    virtual uint32_t Compute(CpuKernelContext &ctx) override;    // 声明函数Compute，且Compute函数需要
重写
};
} // namespace aicpu
#endif
```

- 引入相关头文件。
  - 头文件cpu_kernel.h，其包含了AI CPU算子基类CpuKernel的定义，以及Kernels的注册宏的定义。
  - 头文件cpu_tensor.h，包含了AI CPU的Tensor类的定义及相关方法。
  - 头文件cpu_tensor_shape.h，包含了AI CPU的TensorShape类及相关方法。
  - 头文件cpu_types.h，包含了AI CPU的数据类型以及格式等定义。
  - 头文件cpu_attr_value.h，包含了包含了AttrValue类的属性定义以及方法。
- 进行算子类的声明，此类为CpuKernel类的派生类，并需要声明重载函数Compute，Compute函数需要在算子实现文件中进行实现，详细请参见**源文件代码模块介绍**。算子类的声明需要在命名空间“**aicpu**”中，命名空间的名字“aicpu”为固定值，不允许修改。

源文件代码模块介绍

用户需要在算子工程的“cpukernel/impl/reshape_cust_kernel.cc”文件中进行算子的计算逻辑实现，代码模块介绍如下所示：

```
#include "reshape_cust_kernel.h"     // 引入声明ReshapeCust算子类的头文件

namespace {
const char *RESHAPE_CUST = "ReshapeCust";      //算子的OpType为ReshapeCust
}

namespace aicpu {    // 定义命名空间aicpu
uint32_t ReshapeCustCpuKernel::Compute(CpuKernelContext &ctx)  // 实现自定义算子类的Compute函数
```

```
{
    return 0;
}

REGISTER_CPU_KERNEL(RESHAPE_CUST, ReshapeCustCpuKernel); // 注册ReshapeCust算子实现
} // namespace aicpu
```

1. 引入相关头文件。

   头文件*reshape_cust_kernel.h*，**头文件代码模块介绍**中声明的头文件。

   以下头文件，若在**头文件代码模块介绍**已经引入，无需在重复引入。

   - 头文件cpu_kernel.h，其包含了AI CPU算子基类CpuKernel的定义，以及Kernels的注册宏的定义。
   - 头文件cpu_tensor.h，包含了AI CPU的Tensor类的定义及相关方法。
   - 头文件cpu_tensor_shape.h，包含了AI CPU的TensorShape类及相关方法。
   - 头文件cpu_types.h，包含了AI CPU的数据类型以及格式等定义。
   - 头文件cpu_attr_value.h，包含了包含了AttrValue类的属性定义以及方法。

2. 定义命名空间，声明常量字符指针指向算子的OpType。

   如下所示：

   ```
   namespace {
   const char *RESHAPE_CUST = "ReshapeCust";
   }
   ```

   其中，ReshapeCust为算子的OpType，*RESHAPE_CUST*为声明的指向算子OpType的常量指针。

3. 定义命名空间**aicpu**，并在命名空间aicpu中实现自定义算子的Compute函数，定义算子的计算逻辑。

   命名空间的名称aicpu为固定值，基类及相关定义都在aicpu命名空间中。

   - Compute函数声明。

     ```
     uint32_t ReshapeCustCpuKernel::Compute(CpuKernelContext &ctx)
     ```

     ReshapeCustCpuKernel为头文件中定义的自定义算子类，形参CpuKernelContext为CPU Kernel的上下文，包括算子的输入输出Tensor以及属性等相关信息。

   - Compute函数体中，根据算子开发需求，编写相关代码实现获取输入tensor相关信息，并根据输入信息组织计算逻辑，得出输出结果，并将输出结果设置到输出tensor中。

4. 注册算子的Kernel实现。

   ```
   REGISTER_CPU_KERNEL(RESHAPE_CUST, ReshapeCustCpuKernel);
   ```

   - 第一个参数*SAMPLE*为**2**中定义的指向算子OpType的字符串指针。
   - 第二个参数*ReshapeCustCpuKernel*为自定义算子类的名称。

Compute 函数实现

AI CPU算子实现的关键代码是Compute的实现，ReshapeCust算子的功能是将输入tensor的数据拷贝到输出tensor中，输出的tensor的shape信息将在算子原型定义的Infershape中进行推导。算子实现代码示例如下所示：

```
namespace aicpu {
uint32_t ReshapeCustCpuKernel::Compute(CpuKernelContext &ctx)
{
    Tensor *input_tensor = ctx.Input(0);    // 可根据获取到的输入input_tensor获取输入的shape,数据等信息
    if (input_tensor == nullptr) {
        return -1;
    }
```

```
    Tensor *output_tensor = ctx.Output(0);    // 可根据获取到的输出output_tensor获取输出的shape,数据等信息
    if (output_tensor == nullptr) {
        return -1;
    }

    // 获取输入tensor的数据地址
    auto input_data = input_tensor->GetData();
    if (input_data == nullptr) {
        return -1;
    }

    // 获取输出tensor的数据地址
    auto output_data = output_tensor->GetData();
    if (output_data == nullptr) {
        return -1;
    }

    // 将输入tensor的数据拷贝至输出tensor中
    uint64_t data_size = input_tensor->GetDataSize();
    memcpy(output_data, input_data, data_size);
    return 0;
}
```

8.8.2 算子原型定义

进入算子工程的“op_proto”目录，编写IR实现文件“reshape_cust.h”和“reshape_cust.cpp”，将算子注册到算子原型库中。网络运行时，GE会调用算子原型库的校验接口进行基本参数的校验，校验通过后，会根据原型库中的推导函数推导每个节点的输出shape与dtype，进行输出tensor的静态内存的分配。

reshape_cust.h 实现

MindStudio已在reshape_cust.h文件中生成了算子注册的代码模板文件，开发者可根据需要进行修改，ReshapeCust算子的原型定义如下所示：

```
#ifndef GE_OP_INTERP_RESHAPE_CUST_H
#define GE_OP_INTERP_RESHAPE_CUST_H

#include "graph/operator_reg.h"
namespace ge {
REG_OP(ReshapeCust)
    .INPUT(tensor, TensorType({DT_BOOL, DT_FLOAT16, DT_FLOAT, DT_INT8, DT_INT32, DT_UINT32,
DT_UINT8,
                        DT_INT64, DT_UINT64, DT_INT16, DT_UINT16, DT_DOUBLE, DT_COMPLEX64,
                        DT_COMPLEX128, DT_QINT8, DT_QUINT8, DT_QINT16, DT_QUINT16, DT_QINT32}))
    .INPUT(shape, TensorType({DT_INT32, DT_INT64}))
    .OUTPUT(output, TensorType({DT_BOOL, DT_FLOAT16, DT_FLOAT, DT_INT8, DT_INT32, DT_UINT32,
DT_UINT8,
                         DT_INT64, DT_UINT64, DT_INT16, DT_UINT16, DT_DOUBLE, DT_COMPLEX64,
                         DT_COMPLEX128, DT_QINT8, DT_QUINT8, DT_QINT16, DT_QUINT16, DT_QINT32}))
    .OP_END_FACTORY_REG(ReshapeCust)

}
#endif // GE_OP_INTERP_RESHAPE_CUST_H
```

- REG_OP(ReshapeCust)中的Add为算子注册到昇腾AI处理器中的Type，需要与**8.8.4 算子适配插件实现**中REGISTER_CUSTOM_OP("ReshapeCust")中的算子类型保持一致。
- .INPUT与.OUTPUT分别为算子的输入、输出Tensor的名称与数据类型，输入输出的顺序需要与**8.8.1 算子代码实现**函数形参顺序以及**算子信息定义**中参数的顺序保持一致。

reshape_cust.cpp 实现

原型定义的关键点是推理输出Tensor的shape。ReshapeCust算子推理输出shape的原理为：首先获取两个输入，一个是输入tensor，一个是目标的shape，校验输入tensor元素个数是否与目标shape相同，校验成功则将输出shape设置为目标shape值。

MindStudio已在reshape_cust.cpp文件中生成了代码模板文件，开发者可根据需要进行修改。reshape_cust.cpp的实现代码如下所示：

```
#include "reshape_cust.h"          //IR注册头文件
#include <vector>        //可使用vector类模板并调用vector相关接口。
#include <string>        //C++标准库，可使用string类构造对象并调用string相关接口
#include <iostream>       //C++标准库，调用输入输出流接口

namespace {
// 获取目标shape
template <typename T>
std::vector<int64_t> AsInt64(const T *data, int64_t data_size) {
  std::vector<int64_t> ret(data_size);
  for (int64_t i = 0; i < data_size; ++i) {
    ret[i] = data[i];
  }
  return ret;
}

// 根据shape获取元素个数
int64_t GetElementNum(const std::vector<int64_t> &shape) {
  int64_t ret = 1;
  for (size_t i = 0; i < shape.size(); ++i) {
    ret *= shape[i];
  }
  return ret;
}
}

namespace ge {

IMPLEMT_COMMON_INFERFUNC(ReshapeCustInferShape) {
  TensorDesc tensordesc_tensor = op.GetInputDesc("tensor");
  TensorDesc tensordesc_shape = op.GetInputDesc("shape");
  TensorDesc tensordesc_output = op.GetOutputDesc("output");
  Tensor shape_tensor;
  // 获取目标shape的值
  if (op.GetInputConstData("shape", shape_tensor) == GRAPH_SUCCESS) {
    DataType shape_type = tensordesc_shape.GetDataType();
    std::vector<int64_t> shape_values;
    if (shape_type == DT_INT32) {
      auto shape_data = reinterpret_cast<const int32_t *>(shape_tensor.GetData());
      shape_values = AsInt64<int32_t>(shape_data, shape_tensor.GetSize());
    } else {
      auto shape_data = reinterpret_cast<const int64_t *>(shape_tensor.GetData());
      shape_values = AsInt64<int64_t>(shape_data, shape_tensor.GetSize());
    }

    // 校验目标shape是否合法
    std::vector<int64_t> input_shape = tensordesc_tensor.GetShape().GetDims();
    int64_t input_element_num = GetElementNum(input_shape);
    int64_t shape_element_num = GetElementNum(shape_values);
    if (input_element_num != shape_element_num) {
      return GRAPH_FAILED;
    }

    // 设置输出tensor的shape
    tensordesc_output.SetShape(Shape(input_shape));
    tensordesc_output.SetOriginShape(Shape(input_shape));
  }

  tensordesc_output.SetDataType(tensordesc_tensor.GetDataType());
```

```
  std::vector<std::pair<int64_t,int64_t>> range;
  auto status = op.GetInputDesc("tensor").GetShapeRange(range);
  if (status != GRAPH_SUCCESS) {
    return GRAPH_FAILED;
  }
  tensordesc_output.SetShapeRange(range);

  (void)op.UpdateOutputDesc("output", tensordesc_output);
  return GRAPH_SUCCESS;
}

COMMON_INFER_FUNC_REG(ReShapeCust, ReshapeCustInferShape);

}
```

8.8.3 算子信息定义

原理

算子信息库作为算子开发的交付件之一，主要体现算子在昇腾AI处理器上的具体实现规格，包括算子支持输入输出type、name等信息。网络运行时，根据算子信息库中的算子信息做基本校验，并进行算子匹配。

配置说明

进入“cpukernel/op_info_cfg/aicpu_kernel”目录，配置算子信息库文件“reshape_cust.ini”。开发者需要基于MindStudio自动生成的reshape_cust.ini文件进行修改，修改后的ReshapeCust算子的算子信息定义如下所示。

```
[ReshapeCust]
opInfo.engine=DNN_VM_AICPU
opInfo.flagPartial=False
opInfo.computeCost=100
opInfo.flagAsync=False
opInfo.opKernelLib=CUSTAICPUKernel
opInfo.kernelSo=libcust_aicpu_kernels.so
opInfo.functionName=RunCpuKernel
opInfo.workspaceSize=1024
```

参数说明请参见**表8-24**。下表仅列出常用的算子信息库配置项，其他配置项请参见以下手册中的算子信息库定义章节。

- 《**AI CPU自定义算子开发指南(推理)**》
- 《**AI CPU自定义算子开发指南(训练)**》

表 8-24 ReshapeCust 算子的算子信息定义说明

信息	ReshapeCust算子配置	说明
[OpType]	[ReshapeCust]	算子类型，以英文半角范括号，标识一个算子信息开始，根据**8.3 算子分析**，算子类型为ReshapeCust。
opInfo.engine	DNN_VM_AICPU	配置算子调用的引擎。 AI CPU自定义算子的引擎固定为“DNN_VM_AICPU”。

opInfo.flagPartial	False	此字段为预留字段，请保持固定值“False”。
opInfo.computeCost	100	此字段为预留字段，请保持固定值“100”。
opInfo.flagAsync	False	此字段为预留字段，请保持固定值“False”。
opInfo.opKernelLib	CUSTAICPUKernel	配置算子调用的kernelLib。 AI CPU自定义算子调用的kernelLib固定为“CUSTAICPUKernel”。
opInfo.kernelSo	libcust_aicpu_kernels.so	配置AI CPU算子编译生成的so的名称。
opInfo.functionName	RunCpuKernel	配置自定义算子调用的kernel函数接口名称。 自定义算子的kernel函数接口固定为“RunCpuKernel”。
opInfo.workspaceSize	1024	此字段为预留字段。 配置为内存空间，用于分配算子临时计算的内存。 单位为KB，取值范围为：0~1048576（1G）。 建议配置为：1024

8.8.4 算子适配插件实现

开发者需要进行算子适配插件的开发，实现将Tensorflow网络中的算子进行解析并映射成昇腾AI处理器中的算子。MindStudio在“framework/tf_plugin/tensorflow_reshape_cust_plugin.cpp”文件已自动生成了ReshapeCust算子的插件代码。

- 包含头文件。
```
//包含该头文件，可使用算子注册类相关，调用算子注册相关的接口，为Ascend-cann-toolkit开发套件包中atc/include/register/register.h文件
#include "register/register.h"
```
- 进行插件注册。
```
using namespace ge;    // 需手动添加
namespace domi {
// register op info to GE
REGISTER_CUSTOM_OP("ReshapeCust")
    .FrameworkType(TENSORFLOW)   // type: CAFFE, TENSORFLOW
    .OriginOpType("ReshapeCust")      // name in tf module
    .ParseParamsByOperatorFn(AutoMappingByOpFn);
    .ImplyType(ImplyType::AI_CPU);    // 需手动添加
}  // namespace domi
```
  - REGISTER_CUSTOM_OP：算子注册到GE的算子类型，根据算子分析，算子类型为“ReshapeCust”。
  - FrameworkType：TENSORFLOW，原始框架类型为TensorFlow。

- OriginOpType：算子在TensorFlow框架中的类型。
- ParseParamsByOperatorFn：用来注册解析模型的函数，使用AutoMappingFn函数自动实现解析。
- ImplyType：指定算子的实现方式，ImplyType::AI_CPU表示该算子是AI CPU算子。需要手动添加。

8.8.5 算子工程编译部署

AI CPU算子工程编译部署，请用户参考

- 《**AI CPU自定义算子开发指南(推理)**》
- 《**AI CPU自定义算子开发指南(训练)**》

手册中的“算子工程编译部署”小节。

8.8.6 ST 测试

AI CPU算子ST测试，请用户参考

- 《**AI CPU自定义算子开发指南(推理)**》
- 《**AI CPU自定义算子开发指南(训练)**》

手册中的“算子ST测试”章节。

8.9 其他功能及操作

8.9.1 算子工程管理

8.9.1.1 导入算子工程样例

MindStudio工具提供了完整的算子实现代码样例，可以参考本节内容导入算子工程样例，无需修改算子代码即可体验算子验证、算子部署、算子网络验证过程。

步骤1 进入工程导入页面。

- 首次登录MindStudio：在**Welcome to MindStudio**窗口中，单击**Open or Import**。
- 非首次登录MindStudio：在顶部菜单栏选择“File > Open...”。

步骤2 在**Open File or Project**窗口中，选择样例工程所在文件夹。

TBE样例工程所在路径为MindStudio安装路径下的“MindStudio/samples/demo_projects/custom/”。

步骤3 单击**OK**，导入TBE样例工程。

若打开工程时出现如图8-29所示提示，是因为样例工程配置文件中未配置ADK的版本号，需要单击**OK**修改为当前实际使用的ADK版本号，如图8-30所示。

图 8-29 工程打开异常提示

图 8-30 工程属性修改

----结束

8.9.1.2 修改算子工程属性

创建完算子工程后，支持修改工程属性，包括ADK版本以及产品形态，方便用户通过一个工程运行不同的产品形态。

前提条件

已经创建工程，算子工程创建详细操作请参见**8 自定义算子开发**。

操作步骤

步骤1 选中已经创建的算子工程，右击选择“Edit Operators Property”，弹出**图8-31**所示界面。

图 8-31 工程属性修改

界面参数解释如**表8-25**所示。

表 8-25 工程属性修改界面参数

参数	说明
Name	工程名称，创建算子工程时决定，不支持修改。
Description	工程描述信息，支持修改。
ADK Version	当前ADK的版本号。
Project Location	工程的存储路径，创建算子工程时决定，不支持修改。

步骤2 根据实际情况修改“ADK Version”，修改完后，单击“OK”。进行其他工程开发。

----结束

8.9.1.3 切换到算子逻辑视图

您可以将前算子工程视图切换为算子逻辑视图，该视图按照算子维度展示算子工程目录下的文件，实际文件结构和位置不发生变化。

您可以在算子逻辑视图下进行算子开发、算子验证、并且能方便地新增和删除算子。

在算子工程的Project窗口中，单击 按钮，在下拉框中选择“Operators”，如图8-32所示。

图 8-32 切换入口

切换成功后，可以查看算子逻辑视图，如图8-33所示。

图 8-33 查看算子逻辑视图

该视图以算子维度展示算子交付件，包括算子信息定义文件、算子实现文件、算子原型定义文件、算子插件实现文件、算子融合信息定义文件、算子测试用例。

8.9.1.4 新增/删除算子

本节介绍如何在已创建的工程中添加新算子，删除已有算子。

在算子工程视图中新增算子

在算子工程视图中，选中算子工程，右键选择“New > Operator”，可以在弹出框中设置新算子信息，如图8-34所示。从而实现在当前算子工程中实现多个算子，如果算子类型重复，会提示用户是否进行覆盖。

图 8-34 在算子工程视图中新增算子

在算子逻辑视图中新增算子

8.9.1.3 切换到算子逻辑视图，选中算子工程，右键选择“New > Operator”，可以在弹出框中新增算子信息，如图8-35所示。如果算子类型重复，会提示用户是否进行覆盖。

图 8-35 在算子逻辑视图中新增算子

在算子逻辑视图中删除算子

8.9.1.3 切换到算子逻辑视图，选中待删除的算子名称，右键选择“Delete Operator”，如图8-36所示。

图 8-36 在算子逻辑视图中删除算子

8.9.2 运行生成*.cce 文件

分析算子*.cce文件，可以了解到底层实现的数据搬运过程、buffer切分等，用以优化算子实现。

正常情况下，算子运行后，生成算子二进制文件*.o和算子描述文件*.json，如果还需要生成*.cce文件，请在算子运行前编辑***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux/atc/python/site-packages/te/platform/cce_build.py**文件，配置save_temp_cce_file为True。

算子运行后，在算子二进制文件*.o和算子描述文件*.json同级目录下，生成*.cce文件。

8.9.3 TIK 功能调试

前提条件

已完成TIK算子实现代码。

操作步骤

本章节以simple_add算子为例介绍TIK功能调试方法。

步骤1 编写simple_add算子调试代码.py文件。

完整代码示例参考：（以文件名simple_add.py为例 ）

```
import numpy as np

from te import tik
from te.platform.cce_conf import te_set_version

def simple_add():
    tik_instance = tik.Tik()
    kernel_name = "tik_vec_add_128_float32"
    dst_ub = tik_instance.Tensor("float32", [128], tik.scope_ubuf, "dst_ub")
    dst_gm = tik_instance.Tensor("float32", (128,), tik.scope_gm,  "dst_gm")
    src0_gm = tik_instance.Tensor("float32", (128,), tik.scope_gm, "src0_gm")
    src0_ub = tik_instance.Tensor("float32", (128,), tik.scope_ubuf, "src0_ub")
    src1_gm = tik_instance.Tensor("float32", (128,), tik.scope_gm, "src1_gm")
    src1_ub = tik_instance.Tensor("float32", (128,), tik.scope_ubuf, "src1_ub")

    tik_instance.data_move(src0_ub, src0_gm, 0, 1, 16, 0, 0)
    tik_instance.data_move(src1_ub, src1_gm, 0, 1, 16, 0, 0)
    tik_instance.vec_add(64, dst_ub, src0_ub, src1_ub, 2, 8, 8, 8)
```

```
    tik_instance.data_move(dst_gm, dst_ub, 0, 1, 16, 0, 0)
    tik_instance.BuildCCE(kernel_name, [src0_gm, src1_gm], [dst_gm])

    return tik_instance

if __name__=='__main__':
    te_set_version("Ascend310")   # 请根据芯片实际版本设置，具体请参考te_set_version接口说明
    tik_instance = simple_add()
    data_x = np.ones((128,)).astype("float32")
    data_y = np.ones((128,)).astype("float32")
    feed_dict = {'src0_gm': data_x, 'src1_gm': data_y}
    model_data, = tik_instance.tikdb.start_debug(feed_dict=feed_dict,interactive=True)
    print(model_data)
```

步骤2 右键单击simple_add.py文件选择“RUN”或者通过单击界面上的 图标，开始调试TIK算子。

详细操作请参考《**TBE自定义算子开发指南(推理)**》或《**TBE自定义算子开发指南(训练)**》的“算子开发过程>算子代码实现（TIK开发方式）>功能调试”章节。

----结束

9 精度比对

9.1 简介与约束
9.2 比对数据准备
9.3 Vector比对
9.4 FAQ

9.1 简介与约束

简介

在模型转换过程中对模型进行了优化，包括算子消除、算子融合、算子拆分，可能会造成自有实现的算子运算结果与业界标准算子（如Caffe、TensorFlow、ONNX）运算结果存在偏差，此时需要提供工具比对两者之间的差距，帮助开发人员快速解决算子精度问题。

精度比对工具提供比对华为自有模型算子的运算结果与Caffe、TensorFlow、ONNX标准算子的运算结果，以便确认误差发生的算子，目前提供以下比对项目：余弦相似度、最大绝对误差、累积相对误差、欧氏相对距离、KLD散度、标准差的算法比对。

参考手册

表 9-1 参考手册

产品	手册
Atlas 200 DK	《昇腾模型压缩工具使用指南（Caffe）》
其他产品	《昇腾模型压缩工具使用指南（Caffe）》

精度比对场景

当前版本，精度比对支持通过昇腾AI处理器运行生成的dump数据与Ground Truth（基于GPU/CPU运行生成的dump数据）进行比对，同时还支持量化、非量化的dump数据比对，具体范围如表9-2所示。

表 9-2 精度比对场景

序号	待比对数据（My Output）	标准数据（Ground Truth）
推理场景		
1	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(Caffe)
2	量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(Caffe)
3	量化原始模型的npy文件(Caffe)	非量化原始模型的npy文件(Caffe)
4	量化离线模型在昇腾AI处理器上运行生成的dump数据	量化原始模型的npy文件(Caffe)
5	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(TensorFlow)
6	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(ONNX)
训练场景		
1	通过昇腾AI处理器运行生成的训练网络dump数据	TensorFlow原始训练网络dump数据

使用约束

精度比对支持的dump数据的Format类型：

- NCHW
- NHWC
- ND
- NC1HWC0
- FRACTAL_Z
- HWCN

精度比对支持的dump数据的类型：

- FLOAT
- FLOAT16
- DT_INT8
- DT_UINT8

- DT_INT16
- DT_UINT16
- DT_INT32
- DT_INT64
- DT_UINT32
- DT_UINT64
- DT_BOOL
- DT_DOUBLE

精度比对前环境准备

本章以HwHiAiUser普通用户安装为例，介绍精度比对的操作方法，请实际操作时根据您自己的环境进行替换。

精度比对工具需要配套python3.7.5版本使用。

9.2 比对数据准备

9.2.1 比对数据格式要求

当前版本支持多种比对方式，因此，对dump、npy数据文件命名有以下明确要求，请您准备数据时务必遵循。

表 9-3 数据文件命名规则

dump数据类型	dump数据命名格式
非量化原始模型的dump数据(Caffe)	{op_name}.{output_index}.{timestamp}.pb
量化原始模型的dump数据(Caffe)	{op_name}.{output_index}.{timestamp}.quant
非量化离线模型在昇腾AI处理器上运行生成的dump数据	{op_type}.{op_name}.{task_id}.{timestamp}
量化离线模型在昇腾AI处理器上运行生成的dump数据	{op_type}.{op_name}.{task_id}.{timestamp}
非量化原始模型的dump数据(TensorFlow)	{op_name}.{output_index}.{timestamp}.pb
npy数据(Caffe或TensorFlow或ONNX)	{op_name}.{output_index}.{timestamp}.npy

命名格式说明：op_type、op_name对应的名称需满足“A-Za-z0-9_-”正则表达式规则，timestamp为16位时间戳，output_index、task_id为0~9数字组成。

9.2.2 推理场景数据准备

9.2.2.1 准备离线模型 dump 数据

本节介绍通过MindStudio提供的dump功能，生成离线模型的dump数据。请在启动dump前，您需要已完成开发环境准备、并完成应用工程开发、编译和运行，确保具备可执行的应用工程。

您可以参考以下步骤生成dump数据：

步骤1 打开MindStudio，选择“Ascend > Dump Configuration”菜单，弹出“Select Offline Model”窗口，如图9-1所示。

说明

您也可以通过打开工程文件，然后右键单击.om模型文件，选择“Dump Configuration”菜单的方式配置dump。

图 9-1 选择.om 模型文件

步骤2 选择.om模型文件，单击“Open”，展示模型文件结构，设置dump开关。如图9-2所示。

说明

- 如果涉及多个模型需要dump数据，则需要分别设置各个模型文件的dump配置项。
- 当使用相同系列网络模型时，可能会存在因模型的Model Name值相同，造成生成的acl.json文件中dump配置混淆。

  例如，先使用yolov2进行了dump配置，然后又使用yolov3，此时虽然未对yolov3配置，但acl.json文件中已经记录了yolov2的dump配置，又因为yolov2与yolov3的Model Name值相同，这样就会出现yolov3继承了yolov2的配置。

  因此，当存在因使用相同系列网络模型造成Model Name相同时，可以通过修改Caffe模型文件prototxt中的name字段值并重新进行模型转换，使得系列模型的name字段值不同；也可以在dump配置时先选择None进行清理之前的配置，再次重新进行dump配置。

图 9-2 设置 dump

通过窗口右侧配置项，设置.om模型文件的dump配置项。

- Dump Option：配置dump范围。
  - ALL：所有算子开启dump。

    说明

    不具有输出的TBE算子、AI CPU算子，如StreamActive、NetOutput、Send、Recv、const等不会生成dump数据。
  - Several：自定义部分算子开启dump。

    选择该项后，需要右键单击待dump数据的算子并选择“Enable Dump”，此时算子变为橙色底纹。

说明

对于关联的算子需要同时开启或关闭dump：通过ATC解析om模型文件得到算子之间的映射关系，识别出是新增算子、一对多、多对一、多对多、L1 Batch等关系。

const算子不支持开启dump。

采用dump部分算子场景下，因data算子不会在AI CPU或AI Core上执行，如果用户dump data节点算子，需要一并选择dump data节点算子的后继节点，才能dump出data节点算子数据。

不具有输出的TBE算子、AI CPU算子，如StreamActive、Send、Recv、const等不会生成dump数据；编译后的模型中部分算子并不会在AI CPU或AI Core执行，如concatD类型算子，则无法生成dump数据。

  - None：所有算子不开启dump。
- Dump Mode：dump数据模式。
  - Input：dump算子的输入数据。
  - Output：dump算子的输出数据。
  - All：同时dump算子的输入、输出数据。
- Dump Path：配置保存dump数据文件的路径，默认为：{project_path}/dump。如果Dump Path设置为其他路径，需要确保MindStudio安装用户对该路径具有读写权限。

设置完dump配置后，算子的dump配置信息保存到{project_path}/src/acl.json文件，格式如图9-3或图9-4所示。当选择整个模型dump时，不含layer字段。

图 9-3 整网所有算子开启 dump

```
1  {
2      "dump":{
3          "dump_list":[
4              {
5                  "model_name":"ResNet-50"
6              }
7          ],
8          "dump_mode":"input",
9          "dump_path":"/home/user/AscendProjects/MyApp/dump"
10     }
11 }
```

图 9-4 部分算子开启 dump

```
1  {
2      "dump":{
3          "dump_list":[
4              {
5                  "model_name":"ResNet-50",
6                  "layer":[
7                      "pool1",
8                      "res2a_branch2ares2a_branch2a_relu",
9                      "conv1conv1_relu"
10                 ]
11             }
12         ],
13         "dump_mode":"input",
14         "dump_path":"/home/user/AscendProjects/MyApp/dump"
15     }
16 }
17
```

步骤3 设置dump完成后，分别单击MindStudio界面“Build”和“Run”菜单，重新编译和运行应用工程。

说明

建议先检查工程文件代码，确保正确引用了设置dump配置的.om模型文件。例如检查aclmdlLoadFromFile()函数或aclmdlLoadFromFileWithMem()函数。

工程运行完毕后，可以在{project_path}/dump路径下查看到生成的dump数据文件。生成的路径及格式说明：

```
time/device_id/model_name/model_id/data_index/dump文件
```

- time：dump数据回传落盘时间。格式为：YYYYMMDDhhmmss。
- device_id：Device设备ID号。
- model_name：模型名称。
- model_id：模型ID号。
- data_index：针对每个Task ID执行的次数维护一个序号，从0开始计数，该Task每dump一次数据，序号递增1。
- dump文件：命名规则如{op_type}.{op_name}.{taskid}.{timestamp}。

如果model_name、op_type、op_name出现了“.”、“/”、“\”、空格时，转换为下划线表示。

----结束

9.2.2.2 准备 Caffe 模型 npy 数据文件

本版本不提供Caffe模型npy数据生成功能，请自行安装Caffe环境并提前准备Caffe原始数据“*.npy”文件。我们仅提供生成符合精度比对要求的numpy格式Caffe原始数据“*.npy”文件的样例参考。

说明

如何准备原始caffe模型npy数据，您可以参考论坛发帖**算子精度比对工具标杆数据生成环境搭建指导（Caffe + TensorFlow）**或者自行获取其他方法。该贴仅供参考。

如果您需要使用二进制格式dump文件进行比对，请参考**9.4.3 如何进行dump数据文件与numpy文件格式互转**完成npy文件转换成dump文件。

Caffe原始数据文件准备要求：

- 文件内容以 numpy格式保存。
- 文件命名以op_name.output_index.timestamp.npy形式命名。设置numpy数据文件名包括output_index字段且值为0，确保转换生成的dump数据的output_index为0。因为，精度比对时默认从第一个output_index为0的数据开始，否则无比对结果。
- 为确保生成符合命名要求的.npy文件，需要对原始的Caffe模型文件去除in-place，生成新的.prototxt模型文件用于生成.npy文件（例如：如果有未去除in-place的A、B、C、D四个融合算子，进行dump数据，输出的结果为D算子的结果，但命名却是A算子开头，就会导致比对时找不到文件）。针对量化场景，需要先在环境上安装昇腾模型压缩工具再执行去除in-place命令，安装方法请参考**表9-1**中的《昇腾模型压缩工具使用指南（Caffe）》。

  进入***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux/toolkit/tools/operator_cmp/compare**目录，执行命令去除in-place，命令行举例如下：

  python3.7.5 inplace_layer_process.pyc -i /home/HwHiAiUser/resnet50.prototxt

  执行命令后，在/home/HwHiAiUser目录下生成去除in-place的new_resnet50.prototxt文件。
- 针对量化场景：为确保精度误差，需要执行Caffe模型推理时预处理数据与Caffe模型小型化时预处理数据一致。

为输出符合精度比对要求的“*.npy”数据文件，需在推理结束后的代码中增加以下类似代码：

```
#read prototxt file
net_param = caffe_pb2.NetParameter()
with open(self.model_file_path, 'rb') as model_file:
    google.protobuf.text_format.Parse(model_file.read(), net_param)

    # save data to numpy file
    for layer in net_param.layer:
        name = layer.name.replace("/", "_").replace(".", "_")
        index = 0
        for top in layer.top:
            data = net.blobs[top].data[...]
            file_name = name + "." + str(index) + "." + str(
                round(time.time() * 1000000)) + ".npy"
            output_dump_path = os.path.join(self.output_path, file_name)
            np.save(output_dump_path, data)
            os.chmod(output_dump_path, FILE_PERMISSION_FLAG)
            print('The dump data of "' + layer.name
                  + '" has been saved to "' + output_dump_path + '".')
            index += 1
```

增加上述代码后，运行Caffe模型的应用工程，即可生成符合要求的“*.npy”数据文件。

9.2.2.3 准备 TensorFlow 模型 npy 数据文件

本版本不提供TensorFlow模型npy数据生成功能，请自行安装TensorFlow环境并提前准备npy数据。我们仅提供生成numpy格式TensorFlow原始数据“*.npy”文件的样例参考。

说明

如果您需要使用二进制格式dump文件进行比对，请参考**9.4.3 如何进行dump数据文件与numpy文件格式互转**完成npy文件转换成dump文件。

在进行TensorFlow模型生成npy数据前，您需要已经有一套完整的、可执行的、标准的TensorFlow模型应用工程。然后利用TensorFlow官方提供的debug工具tfdbg调试程序，从而生成npy文件。主要操作示例如下，请根据自己的应用工程适配操作：

步骤1 修改TF应用工程脚本，添加debug选项设置。代码中增加如下代码：

estimator模式：

from tensorflow.python import debug as tf_debug

training_hooks = [train_helper.PrefillStagingAreaHook(), tf_debug.LocalCLIDebugHook()]

session.run模式：

from tensorflow.python import debug as tf_debug

sess = tf_debug.LocalCLIDebugWrapperSession(sess, ui_type="readline")

- estimate模式，如**图9-5**所示示例，添加tfdbg的hook。

图 9-5 estimate 模式

```
from tensorflow.python import debug as tf_debug
def get_classifier(self):
    classifier = tf.estimator.Estimator(
        model_fn=self.model.get_estimator_model_func,
        model_dir=self.config['log_dir'],
        config = tf.estimator.RunConfig(
                session_config=self.sess.get_config(),
                save_summary_steps=self.config['save_summary_steps'] if self.config['do_checkpoint'] else None,
                save_checkpoints_steps=self.config['save_checkpoints_steps'] if self.config['do_checkpoint'] else None,
                keep_checkpoint_max=None
                 )
        )

    #training_hooks = [train_helper.PrefillStagingAreasHook()]
    training_hooks = [train_helper.PrefillStagingAreasHook(), tf_debug.LocalCLIDebugHook()]
    training_hooks.append(self.logger)

    return classifier, training_hooks
```

- session.run模式，如**图9-6**所示示例，在run之前设置tfdbg装饰器。

图 9-6 session.run 模式

```
import numpy as np
import tensorflow as tf
from tensorflow.python import debug as tf_debug

xs = np.linspace(-0.5, 0.49, 100)
x = tf.placeholder(tf.float32, shape=[None], name="x")
y = tf.placeholder(tf.float32, shape=[None], name="y")
k = tf.Variable([0.0], name="k")
y_hat = tf.multiply(k, x, name="y_hat")
sse = tf.reduce_sum((y - y_hat) * (y - y_hat), name="sse")
train_op = tf.train.GradientDescentOptimizer(learning_rate=0.02).minimize(sse)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

sess = tf_debug.LocalCLIDebugWrapperSession(sess,ui_type="readline")
for _ in range(10):
        sess.run(y_hat,feed_dict={x:xs,y:10*xs})
        sess.run(train_op, feed_dict={x: xs, y: 42 * xs})
```

步骤2 执行运行脚本。

进入调试命令行交互模式后，输入run命令。

步骤3 收集npy文件。

执行run命令完成后，在命令行交互界面，可以通过lt查询已存储的张量，通过pt可以查看已存储的张量内容，可以保存数据为numpy格式文件。

因为tfdbg一次命令只能dump一个tensor，为了自动生成收集所有数据，可以按以下几个步骤操作：

1. 执行**lt > *tensor_name***将所有tensor的名称暂存到文件里。
2. 退出tfdbg命令行，在linux命令行下执行下述命令，用以生成在tfdbg命令行执行的命令

timestamp=$[$(date +%s%N)/1000] ; cat tensor_name | awk '{print "pt", $4,$4}' | awk '{gsub("/", "_", $3);gsub(":", ".", $3);print($1,$2,"-n 0 -w "$3"."'"$timestamp'".npy")}'

说明

该示例生成符合精度比对需要的npy文件名称格式。其中，tensor_name为自定义tensor列表对应的文件名，timestamp为16位的时间戳。

本步骤也可以不退出tfdbg命令行，重新开启一个命令窗口，在新的窗口中执行。

3. 回到tfdbg命令行，输入run执行脚本后，将上一步生成的所有tensor存储的命令粘贴执行，即可存储所有npy文件。

   npy文件默认是以numpy.save()形式存储的，上述命令会将“/与:”用下划线_替换。

   说明

   如果命令行界面无法粘贴文件内容，可以在tfdbg命令行中输入“mouse off”指令关闭鼠标模式后再进行粘贴。

4. 检查生成的npy文件命名是否符合规则，如图9-7所示。

说明

- npy文件命名规则：{op_name}.{output_index}.{timestamp}.{npy}，其中op_name字段需满足“A-Za-z0-9_-”正则表达式规则，timestamp为16位时间戳，output_index为0~9数字组成。
- 如果因算子名较长，造成按命名规则生成的npy文件名超过255字符而产生文件名异常，这类算子不支持精度比对。
- 因tfdbg自身原因或运行环境原因，可能存在部分生成的npy文件名不符合精度比对要求，请按命名规则手工重命名。如果不符合要求的npy文件较多，请参考**9.4.5 如何批量处理生成的npy文件名异常情况**重新生成npy文件。

图 9-7 查询.npy 文件

```
truediv_91.0.0123456789012345.npy
truediv_92.0.0123456789012345.npy
truediv_93.0.0123456789012345.npy
truediv_94.0.0123456789012345.npy
truediv_95.0.0123456789012345.npy
truediv_96.0.0123456789012345.npy
truediv_97.0.0123456789012345.npy
truediv_98.0.0123456789012345.npy
truediv_99.0.0123456789012345.npy
```

----结束

9.2.2.4 准备 ONNX 模型 npy 数据文件

本版本不提供ONNX模型npy数据生成功能，请自行安装ONNX环境并提前准备ONNX原始数据“*.npy”文件。我们仅提供生成符合精度比对要求的numpy格式ONNX原始数据“*.npy”文件的样例参考。

ONNX 原始数据文件准备要求

- 文件内容以 numpy格式保存。
- 文件命名以op_name.output_index.timestamp.npy形式命名。设置numpy数据文件名包括output_index字段且值为0，确保转换生成的dump数据的output_index为0。因为，精度比对时默认从第一个output_index为0的数据开始，否则无比对结果。
- 需要确保每个算子节点有名称，否则无法生成正确的文件名。没有名称的算子节点需要先生成名称。
- 本参考示例以每个节点只有一个输出为例，多个输出的需要在文件名生成处适配output_index的获取。

代码参考示例

为输出符合精度比对要求的“*.npy”数据文件，需在推理结束后的代码中增加以下类似代码：

```
import onnx
import onnxruntime
import numpy as np
```

```
import time

from skl2onnx.helpers.onnx_helper import enumerate_model_node_outputs
from skl2onnx.helpers.onnx_helper import select_model_inputs_outputs
from skl2onnx.helpers.onnx_helper import save_onnx_model

#修改模型，增加输出节点
model_onnx = onnx.load("./resnet50.onnx")
output = []
for out in enumerate_model_node_outputs(model_onnx):
    output.append(out)

num_onnx = select_model_inputs_outputs(model_onnx,outputs=output)
save_onnx_model(num_onnx, "resnet50_dump.onnx")

#推理得到输出，本示例中采用随机数作为输入
input_data = np.random.random((1,3,224,224)).astype(np.float32)
input_data.tofile("test_data.bin")
sess = onnxruntime.InferenceSession("resnet50_dump.onnx")
input_name = sess.get_inputs()[0].name
output_name = [node.name for node in sess.get_outputs()]
res = sess.run(output_name, {input_name: input_data})

#获得输出名称，确保每个算子节点有对应名称
node_name = [node.name for node in model_onnx.graph.node]

#保存数据，针对多个输出的节点要适配output_index的循环
for idx,data in enumerate(res):
    file_name = node_name[idx] + "." + str(0) + "." + str(
                    round(time.time() * 1000000)) + ".npy"
    output_dump_path = "./onnx_dump/"+file_name
    np.save(output_dump_path, data.astype(np.float16))
```

9.2.3 训练场景数据准备

9.2.3.1 准备基于昇腾 AI 处理器运行生成的训练网络 dump 数据

前提条件

在进行迁移后的训练网络dump数据前，您需要完成训练网络开发、编译和运行，确保具备可执行的训练工程。

说明

- 如果训练网络包含了随机因子，请在执行生成dump数据前去除。
- 目前仅支持AI CPU和AI Core算子进行dump数据，不支持HCCL等算子dump数据。

生成 dump 数据

参考下面步骤生成网络模型dump数据：

步骤1 修改训练脚本，开启dump功能。

在相应代码中，增加如下的加粗字体信息。

Estimator模式：通过NPURunConfig中的dump_config采集dump数据，在创建NPURunConfig之前，实例化一个DumpConfig类进行dump的配置（包括配置dump路径、dump哪些迭代的数据、dump算子的输入还是输出数据等）。关于DumpConfig类的构造函数中每个字段的详细解释，请参见《**TensorFlow网络模型移植&训练指南**》或《**PyTorch网络模型移植&训练指南**》手册。

```
from npu_bridge.npu_init import *

# dump_path：dump数据存放路径，该参数指定的目录需要在启动训练的环境上（容器或Host侧）提前创建且确保安装时配置的运行用户具有读写权限
# enable_dump：是否开启Dump功能
# dump_step：指定采集哪些迭代的Dump数据
# dump_mode：Dump模式，取值：input/output/all
dump_config = DumpConfig(enable_dump=True, dump_path = "/home/HwHiAiUser/output", dump_step="0|5|10", dump_mode="all")

session_config=tf.ConfigProto()

config = NPURunConfig(
  dump_config=dump_config,
  session_config=session_config
  )
```

sess.run模式：通过session配置项enable_dump、dump_path、dump_step、dump_mode配置dump参数。

```
config = tf.ConfigProto()

custom_op =  config.graph_options.rewrite_options.custom_optimizers.add()
custom_op.name =  "NpuOptimizer"
custom_op.parameter_map["use_off_line"].b = True

# enable_dump：是否开启Dump功能
custom_op.parameter_map["enable_dump"].b = True
# dump_path：dump数据存放路径，该参数指定的目录需要在启动训练的环境上（容器或Host侧）提前创建且确保安装时配置的运行用户具有读写权限
custom_op.parameter_map["dump_path"].s = tf.compat.as_bytes("/home/HwHiAiUser/output")
# dump_step：指定采集哪些迭代的Dump数据
custom_op.parameter_map["dump_step"].s = tf.compat.as_bytes("0|5|10")
# dump_mode：Dump模式，取值：input/output/all
custom_op.parameter_map["dump_mode"].s = tf.compat.as_bytes("all")
config.graph_options.rewrite_options.remapping = RewriterConfig.OFF
config.graph_options.rewrite_options.memory_optimization = RewriterConfig.OFF

with tf.Session(config=config) as sess:
  print(sess.run(cost))
```

步骤2 执行训练脚本，生成dump数据文件。

dump文件存放在{dump_path}/{time}/{deviceid}/{model_name}/{model_id}/{data_index}目录下，以{dump_path}配置/home/HwHiAiUser/output为例，例如存放在“/var/log/npu/ide_daemon/dump/tmp/20200808163566/0/ge_default_20200808163719_121/11/0”目录下。

dump数据文件生成的路径格式说明：

- dump_path：步骤**步骤1**中设置的dump_path路径（如果设置的是相对路径，则为拼接后的全路径）。
- time：dump数据文件落盘的时间。格式为：YYYYMMDDhhmmss。
- deviceid：Device设备ID号。
- model_name：子图名称。model_name层可能存在多个文件夹，dump数据取计算图名称对应目录下的数据。

  执行训练脚本完成后会在训练脚本当前目录生成GE图文件，图文件可能会有多个。一般情况下，选取计算图文件方法：以ge_proto_*****_Build.txt为例，分别打开并查看这些文件，如果包含IteratorV2 或Iterator 或GetNext算子，则为计算图。计算图名称取计算图文件**graph**下的**name**字段值。
- model_id：子图ID号。

- data_index：迭代数，用于保存对应迭代的dump数据。如果指定了dump_step，则data_index和dump_step一致；如果不指定dump_step，则data_index序号从0开始计数，每dump一个迭代的数据，序号递增1。
- dump文件：命名规则如{op_type}.{op_name}.{taskid}.{timestamp}

如果model_name、op_type、op_name出现了“.”、“/”、“\”、空格时，会转换为下划线表示。

说明

- 每次迭代都会产生dump数据，在训练数据集较大的情况下，每次迭代的dump数据量很大（约几十G，甚至更多），建议控制迭代次数，一般仅执行一次迭代。
- 多p环境下，因训练脚本中多device进程调起时间有差异会导致落盘时产生多个时间戳目录。

----结束

9.2.3.2 准备基于 GPU 运行生成的 TensorFlow 原始训练网络 npy 数据文件

前提条件

在进行TensorFlow原始训练网络生成npy或dump数据前，您需要已经有一套完整的、可执行的标准的TensorFlow模型训练工程。

说明

精度比对支持基于昇腾AI处理器运行生成的dump数据与基于GPU/CPU运行生成的npy或dump数据比对，因此，您可以只准备npy文件即可。如果您需要使用基于GPU/CPU运行生成的dump数据比对，请参考**9.4.3 如何进行dump数据文件与numpy文件格式互转**完成npy文件转换成dump文件。

准备 npy 文件

利用TensorFlow官方提供的debug工具tfdbg生成npy文件。详细的操作方法如下：

步骤1 修改tf训练脚本，添加debug选项设置。

- estimate模式，如**图9-8**所示示例，添加tfdbg的hook。

图 9-8 estimate 模式

```
from tensorflow.python import debug as tf_debug
class GPUBaseTrain(object):
    def __init__(self, session, config, data, model, logger):
        self.sess = session
        self.config = config
        self.data = data
        self.model = model
        self.logger = logger
        self.print_logger = self.logger.logger
        self.all_preds = []
        self.all_targets = []
        if self.config['accelerator'] == 'gpu':
            self.classifier, self.training_hook = self.get_classifier()
        else:
           # from tensorflow.contrib.offline_train.python.npu.npu_config import NPURunConfig
            from npu_bridge.estimator.npu.npu_config import NPURunConfig
           # from tensorflow.contrib.offline_train.python.npu.npu_estimator import NPUEstimator
            from npu_bridge.estimator.npu.npu_estimator import NPUEstimator
#            from tensorflow.contrib.offline_train.python.npu.npu_optimizer import NPUDistributedOptimizer
            from npu_bridge.estimator.npu.npu_optimizer import NPUDistributedOptimizer
            self.classifier, self.training_hook = self.get_npu_classifier()


    def get_classifier(self):
        classifier = tf.estimator.Estimator(
            model_fn=self.model.get_estimator_model_func,
            model_dir=self.config['log_dir'],
            config = tf.estimator.RunConfig(
                    session_config=self.sess.get_config(),
                    save_summary_steps=self.config['save_summary_steps'] if self.config['do_checkpoint'] else None,
                    save_checkpoints_steps=self.config['save_checkpoints_steps'] if self.config['do_checkpoint'] else None,
                    keep_checkpoint_max=None
                     )
            )

        training_hooks = [train_helper.PrefillStagingAreasHook(), tf_debug.LocalCLIDebugHook()]
        training_hooks.append(self.logger)
```

- session.run模式，如图9-9所示示例，在run之前设置tfdbg装饰器。

图 9-9 session.run 模式

```
import numpy as np
import tensorflow as tf
from tensorflow.python import debug as tf_debug

xs = np.linspace(-0.5, 0.49, 100)
x = tf.placeholder(tf.float32, shape=[None], name="x")
y = tf.placeholder(tf.float32, shape=[None], name="y")
k = tf.Variable([0.0], name="k")
y_hat = tf.multiply(k, x, name="y_hat")
sse = tf.reduce_sum((y - y_hat) * (y - y_hat), name="sse")
train_op = tf.train.GradientDescentOptimizer(learning_rate=0.02).minimize(sse)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

sess = tf_debug.LocalCLIDebugWrapperSession(sess,ui_type="readline")
for _ in range(10):
        sess.run(y_hat,feed_dict={x:xs,y:10*xs})
        sess.run(train_op, feed_dict={x: xs, y: 42 * xs})
```

步骤2 执行训练脚本。

进入调试命令行交互模式后，输入run命令。

步骤3 收集npy文件。

执行命令完成后，在命令行交互界面，可以通过lt查询已存储的张量，通过pt可以查看已存储的张量内容，可以保存数据为numpy格式文件。

因为tfdbg一次命令只能dump一个tensor，为了自动生成收集所有数据，可以按以下几个步骤操作：

1. 执行**lt > tensor_name**将所有tensor的名称暂存到文件里。
2. 退出tfdbg命令行，在linux命令行下执行下述命令，用以生成在tfdbg命令行执行的命令

timestamp=$[$(date +%s%N)/1000] ; cat tensor_name | awk '{print "pt", $4,$4}' | awk '{gsub("/", "_", $3);gsub(":", ".", $3);print($1,$2,"-n 0 -w "$3"."""$timestamp""".npy")}'

3. 回到tfdbg命令行，输入执行脚本后，将上一步生成的所有tensor存储的命令粘贴执行，即可存储所有npy文件。

   npy文件默认是以numpy.save()形式存储的，上述命令会将“/与:”用下划线_替换。

   说明

   如果命令行界面无法粘贴文件内容，可以在tfdbg命令行中输入“mouse off”指令关闭鼠标模式后再进行粘贴。

4. 检查生成的npy文件命名是否符合规则，如图9-10所示。

说明

- npy文件命名规则：{op_name}.{output_index}.{timestamp}.{npy}，其中op_name字段需满足“A-Za-z0-9_-”正则表达式规则，timestamp为16位时间戳，output_index为0~9数字组成。
- 如果因算子名较长，造成按命名规则生成的npy文件名超过255字符而产生文件名异常，这类算子不支持精度比对。
- 因tfdbg自身原因或运行环境原因，可能存在部分生成的npy文件名不符合精度比对要求，请按命名规则手工重命名。如果不符合要求的npy文件较多，请参考9.4.5 如何批量处理生成的npy文件名异常情况重新生成npy文件。

图 9-10 查询.npy 文件

```
truediv_91.0.0123456789012345.npy
truediv_92.0.0123456789012345.npy
truediv_93.0.0123456789012345.npy
truediv_94.0.0123456789012345.npy
truediv_95.0.0123456789012345.npy
truediv_96.0.0123456789012345.npy
truediv_97.0.0123456789012345.npy
truediv_98.0.0123456789012345.npy
truediv_99.0.0123456789012345.npy
```

----结束

9.3 Vector 比对

9.3.1 比对数据说明

执行Vector比对前，请根据您将要进行Vector比对的场景按照表9-4要求准备好比对数据文件。

说明

My Output离线模型文件与量化融合规则文件使用场景说明：

- 模型文件：使用昇腾AI处理器运行生成的dump数据与Ground Truth比对，选择该模型文件。
- 量化融合规则文件：只要涉及量化与非量化数据比对，则必须选择该文件。

表 9-4 Vector 比对前数据准备

序号	待比对数据（My Output）	标准数据（Ground Truth）	模型文件/融合规则文件
推理场景			
1	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(Caffe)	非量化离线模型文件（*.om）
2	量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(Caffe)	● 量化离线模型文件（*.om） ● 模型小型化后的量化融合规则文件（json文件）
3	量化原始模型的npy文件（或dump数据）(Caffe)	非量化原始模型的npy文件(Caffe)	模型小型化后的量化融合规则文件（json文件）
4	量化离线模型在昇腾AI处理器上运行生成的dump数据	量化原始模型的npy文件(Caffe)	量化离线模型文件（*.om）
5	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(TensorFlow)	非量化离线模型文件（*.om）
6	非量化离线模型在昇腾AI处理器上运行生成的dump数据	非量化原始模型的npy文件(ONNX)	非量化离线模型文件（*.om）
训练场景			
1	通过昇腾AI处理器运行生成的训练网络dump数据	TensorFlow原始训练网络npy文件	计算图文件（*.txt）

9.3.2 比对操作

比对步骤

说明

- 约束：需要确保My Output模型的数据文件与Ground Truth模型的数据文件为相同模型的数据。如果不是相同模型的数据，但模型中有相同的算子名称，也可以进行比较，但结果数据只显示匹配到的相同算子的比对结果。
- 针对FastRCNN网络场景，ProposalD算子及之后的的算子，算子精度不达标属于正常情况，最终结果以FSRDetectionOutput 算子的比对结果精度为准。
- 如果在图编译过程中，原图的算子发生了融合，导致算子的output在编译后的模型中找不到对应的output时，该算子无法进行比对。
- 如果在图编译阶段对图做了结构性修改（如stride切分、L1 fusion、L2 fusion）的场景，会造成算子的input或output无法比对。
- 如果比对数据两边相同算子有dump数据，但算子的shape不一致（离线模型算子shape变小）或format不支持转换，该算子无法进行比对。
- 当模型转换对输入数据做了额外的预处理，造成原始模型的输入与离线模型的data算子输入格式不同时（比如AIPP场景下data输入为YUV），data算子比对结果异常、不具备参考意义。
- 如果没有关闭对应的融合规则，会使得量化算子与之前的算子进行融合，导致该算子的output的比对结果不可信。
- 量化模型里做了量化处理的算子无法比对，必须是反量化的输出才能比对，量化模型里未做量化处理的算子不受影响。例如，量化模型里AscendQuant算子的output无法比对。
- 本节中使用到的文件的存放路径不能包含中文字符、特殊字符等，仅支持双引号中的字符组成的路径：“./\:_- 0-9a-zA-Z”。
- 使用精度比对工具，请确保硬件环境满足要求：CPU 8核 2.6Ghz，内存16GB，否则有可能会造成比对缓慢。

Vector比对操作步骤：

步骤1 在MindStudio界面，在菜单栏选择“Ascend > Model Accuracy Analyzer”菜单，进入比对界面。

步骤2 单击My Output的“Dump Path”右侧的，设置My Output的数据路径，选择npy数据文件或dump数据文件所在的文件夹。如图9-11所示。

图 9-11 设置 My Output 的 Dump Path 参数

步骤3 单击Ground Truth的“Dump Path”右侧的，设置Ground Truth的数据路径，选择npy数据文件或dump数据文件所在的文件夹。如图9-12所示。

图 9-12 设置 Ground Truth 的 Dump Path 参数

步骤4 设置Compare Rule Config配置项，设置离线模型文件、量化融合规则文件。如图9-13所示。

图 9-13 设置 Compare Rule Config

说明

Compare Rule Config配置项由My Output和Ground Truth选择的npy数据文件或dump数据文件的类型决定如何展示，图9-13仅为其中一种展示举例。请根据实际展示的参数项进行设置。

- Model File：选择离线模型文件。
- Model File/Fusion Rule File：推理场景比对选择离线模型文件、训练场景比对选择计算图文件。
- Quant Fusion File：选择量化融合规则文件。

步骤5 （可选）设置显示阈值。

图 9-14 Algorithm

- CosineSimilarity：比对结果页面CosineSimilarity列内容大于等于设定阈值的单元内容标记为绿色，小于阈值的单元标记为红色。界面取值范围[0.00 ~1.00]，默认值：0.99。
- MaxAbsoluteError：指定比对结果页面MaxAbsoluteError列数值最大的前几个数值标记为红色，如果有相同的数值也同时展示红色，其他数值标记为绿色。界面取值范围: [1~5]，默认值：5。
- RelativeEuclideanDistance：比对结果页面RelativeEuclideanDistance列内容小于等于设定阈值的单元内容标记绿色，大于阈值的单元标记为红色。界面取值范围[0.00 ~1.00]，默认值：0.10。

说明

文本框支持手动输入数字，输入数字后按回车键生效。

比对结果出来后，如果修改该处的文本框数值，需要单击对应结果列的排序功能，使修改值生效。

步骤6 单击“Compare”按钮。

结果说明请参考结果展示。

----结束

结果展示

Vector比对结果及说明。

图 9-15 Vector 比对结果

Index	LeftOp	RightOp	TensorIndex	CosineSimilarity	MaxAbsoluteError	AccumulatedRelative...	RelativeEuclideanDist...	KullbackLeiblerDiverg...	StandardDeviation
0	data	data	data:output:0	NaN	151.000000	149610.000000	1.000000	NaN	(0.000;0.000)
110	conv1conv1_relu	conv1 bn_conv1 scale_conv1 conv1_relu	conv1conv1_relu:out...	0.999975	0.181446	9859.678693	0.007113	inf	(1.457;1.635) (1.458;1.636)
111	pool1	pool1	pool1:output:0	0.999982	0.150667	2192.678064	0.006031	inf	(1.859;1.836)
164	pool5	pool5	pool5:output:0	0.996287	1.101688	1079.472061	0.091929	inf	(0.429;0.777)
166	prob	prob	prob:output:0	1.000000	0.001950	997.343049	0.001950	0.000001	(0.000;0.032)

默认仅显示有结果的算子。可单击列名，进行排序。

- LeftOp：表示My Output侧数据的算子名。
- RightOp：表示Ground Truth侧数据的算子名。
- TensorIndex：表示My Output侧算子的Input ID和Output ID。
- Cosine Similarity：进行余弦相似度算法比对出来的结果，范围是[-1,1]，比对的结果如果越接近1，表示两者的值越相近，越接近-1意味着两者的值越相反。
- Max Absolute Error：进行最大绝对误差算法比对出来的结果，值越接近于0，表明越相近，值越大，表明差距越大。
- Accumulated Relative Error：进行累积相对误差算法比对出来的结果，值越接近于0，表明越相近，值越大，表明差距越大。
- Relative Euclidean Distance：进行欧氏相对距离算法比对出来的结果，值越接近于0，表明越相近，值越大，表明差距越大。
- Kullback-Leibler Divergence：进行KLD散度算法比对出来的结果，取值范围是 0到无穷大。KL 散度越小，真实分布与近似分布之间的匹配越好。
- Standard Deviation：进行标准差算法比对出来的结果，取值范围：0到无穷大。标准差越小，离散度越小，表明越接近平均值。该列显示My Output侧dump数据和Ground Truth侧dump数据的均值和标准差，第一组展示My Output侧dump数据的数值(均值;标准差)，第二组展示Ground Truth侧dump数据的数值(均值;标准差)。
- 余弦相似度和KLD散度比较结果为NaN，其他算法有比较数据，则表明左侧或右侧数据为0；KLD散度比较结果为inf，表明右侧数据有一个为0；比对结果为nan，表示dump数据有nan。

说明

- 鼠标悬浮在表头可以看到对应的参数详细解释。
- 点击表格上方“More Help...”链接可以跳转到参数解释的详情页面。此功能需要使用浏览器，如果未安装浏览器，请用户自行安装。

详细比对结果页面

双击Vector比对结果页面某个算子，弹出详细比对结果窗口，如图9-16所示。

说明

单击 按钮，可以快速定位、上下选择Absolute Error列红色背景的数据。

显示详细比对结果的同时，会在MindStudio安装用户家目录下的比对结果路径下生成“{op_name}_{output/input}_{index}_{file_index}.csv”结果文件，例如：~/.mindstudio/compare_tool/result/detail_result/pool1_output_0_0.csv。如果在Windows操作系统下，无法打开该.csv文件，可能原因是文件名过长造成，请修改文件名后重新打开。

图 9-16 详细比对结果

Index	N C H W	Left	Right	Absolute Error	Relative Error
0	0 0 0 0	4.863281	4.864196	0.000915	0.000188
1	0 0 0 1	0.546387	0.546201	0.000186	0.000341
2	0 0 0 2	1.061523	1.061421	0.000103	0.000097
3	0 0 0 3	0.990723	0.990363	0.000360	0.000363
4	0 0 0 4	1.052734	1.052439	0.000295	0.000281
5	0 0 0 5	1.137695	1.137173	0.000522	0.000459
6	0 0 0 6	1.071289	1.071308	0.000019	0.000018
7	0 0 0 7	1.041992	1.041833	0.000159	0.000153
8	0 0 0 8	1.052734	1.052787	0.000053	0.000050
9	0 0 0 9	0.980469	0.980103	0.000366	0.000373
10	0 0 0 10	1.072266	1.072496	0.000231	0.000215
11	0 0 0 11	1.026367	1.026313	0.000054	0.000052
12	0 0 0 12	1.005859	1.005111	0.000749	0.000745
13	0 0 0 13	1.035156	1.034627	0.000529	0.000512
14	0 0 0 14	1.042969	1.042178	0.000791	0.000759
15	0 0 0 15	1.013672	1.013442	0.000230	0.000227
16	0 0 0 16	1.054688	1.054647	0.000041	0.000039
17	0 0 0 17	1.134766	1.134173	0.000593	0.000523
18	0 0 0 18	1.030273	1.029758	0.000515	0.000500
19	0 0 0 19	1.049805	1.049859	0.000054	0.000052
20	0 0 0 20	1.019531	1.018678	0.000853	0.000837
21	0 0 0 21	0.899902	0.899650	0.000252	0.000280
22	0 0 0 22	0.839844	0.839431	0.000413	0.000491
23	0 0 0 23	1.008789	1.008828	0.000039	0.000039
24	0 0 0 24	1.029297	1.029071	0.000226	0.000219

详细比对结果页面数值说明：

- Absolute Error：绝对误差；[0.000000, 0.006815]表示绝对误差的范围；0.004212表示绝对误差的黄金分割点，若下表中的Absolute Error结果大于等于该值，显示为红色，用户可以进行编辑，输入的是整数或float，小数点后最多6位。
- Relative Error：相对误差；[0.000000, 263.258065]表示相对误差的范围；162.693484表示相对误差的黄金分割点，若下表中的Relative Error结果大于等于该值，显示为红色，用户可以进行编辑，输入的是整数或float，小数点后最多6位。
- N C H W：数据格式。
- Left：My Output侧算子的dump值。
- Right：Ground Truth侧算子的dump值。

- Absolute Error：绝对误差，My Output侧算子的dump值减Ground Truth侧算子的dump值取绝对值比对出来的结果。
- Relative Error：相对误差，Absolute Error值除以Ground Truth侧算子的dump值比对出来的结果。当Ground Truth侧算子的dump值为0时，该处显示为“-”。

算子对应关系展示

精度比对完成后，可以单击“Show Model”展示比对算子在模型中的拓扑结构，如图9-17所示。

图 9-17 展示模型结构

在右侧弹出的“Model Compare”窗口中，设置模型文件：

- “My Model”区域对应的“input Model”，选择My Output侧对应的模型文件。
- “Standard Model”区域对应的“input Model”，选择Ground Truth侧对应的模型文件。

设置完成后，MindStudio自动绘制对应的两个模型拓扑结构，如图9-18所示。

- 模型颜色同步余弦相似度算法列的颜色。
- 选择精度比对结果中的某一行，LeftOp和RightOp对应的算子在右侧模型可视化图中高亮显示；选择右侧模型中某一算子，其对应的精度比对结果表中的一行高亮，该行leftOp和rightOp对应的算子也高亮。

图 9-18 模型拓扑结构

9.3.3 子模型导出与比对

简介

在使用昇腾AI处理器的推理功能时，可能存在推理结果和原始模型推理结果不一致的情况，使用精度比对工具可以定位到某个或者某些非用户自定义层出现精度下降的问题，此时需要用户将怀疑有问题的局部网络的相关数据截取出来进行局部结构精度比对，将问题模型最小化后提供给华为工程师定位，在该背景下，MindStudio提供Caffe和TensorFlow原始网络模型的子模型导出功能方便用户获取数据，导出原理如图9-19所示。

图 9-19 子模型导出原理

1. 使用ATC工具，将原始模型转换成适配昇腾AI处理器的离线模型，详情请参见**4 模型转换**。
2. 使用转换后的离线模型进行推理。
3. 如果推理结果和原始模型推理结果不一致，则通过精度比对，将怀疑有问题的原始网络模型的局部网络模型相关数据导出，详情请参见**导出步骤**。重复进行子模型导出与子模型的精度比对，将问题定位至最小模型。
4. 将导出后的数据发给华为工程师定位。

   如果问题解决，则重新进行模型转换，然后进行模型推理。

导出入口

子模型导出有两个入口，用户选择其中一种方式，执行导出操作，本章节以从精度比对界面进入为例进行说明。

- 在精度比对界面，“Dump Path”参数和“Model File”参数选择相关文件后，单击“Dump Network”，进入子模型导出界面，该种方式进入后Dump Data Path参数和Offline Model File参数会自动填充，用户也可以分别单击右侧的自行选择dump文件和离线模型文件。
- 在MindStudio界面，依次单击菜单栏“Ascend > Dump Network”，进入子模型导出界面，该种方式进入后Dump Data Path参数和Offline Model File参数不会自动填充，需要用户分别单击右侧的自行选择dump文件和离线模型文件。

导出步骤

步骤1 参见9.3.2 比对操作完成步骤步骤1~步骤4。

步骤2 “Model File”、“Dump Path”参数选择相关文件后，“Dump Network”变成可编辑状态，在“Model Accuracy Analyzer”精度比对界面，单击“Dump Network”，进入子模型导出界面，如图9-20所示。

图 9-20 子模型导出界面

表 9-5 界面参数以及按钮说明

参数	说明
Dump Data Path	精度比对界面“My Output”区域“Dump Path”中配置的离线模型对应的dump数据，进入“Dump Network”界面后，该参数自行传入Dump Data Path。用户也可以单击右侧的自行选择dump文件。
Offline Model File	精度比对界面“Model File”中配置的离线模型文件，进入“Dump Network”界面后，该参数自行传入Offline Model File。用户也可以单击右侧的自行选择离线模型文件。
Deploy Model	• 原始网络模型文件，单击右侧的选择MindStudio安装服务器相应文件。 • 如果为昇腾模型压缩工具量化后的模型文件，则此处需要选择量化后可在昇腾AI处理器部署的模型文件。

参数	说明
Deploy Weight	● 原始网络模型权重文件，若模型文件与权重文件在MindStudio安装服务器相同位置，选择模型文件后，权重文件会自动填充，否则需要用户单击右侧的 自行选择。 ● 如果为昇腾模型压缩工具量化后的权重文件，则此处需要选择量化后可在昇腾AI处理器部署的权重文件。 **说明** 只有原始网络模型文件为Caffe模型时，才会显示并需要添加Caffe模型权重文件（*.caffemodel）。
Dump Output Path	子模型导出目录，单击右侧的 选择导出目录，若没有目录，请先自行创建。
Show Result	单击该按钮，会解析原始模型文件，并将解析后的结果以图的形式呈现在右侧空白区域。
Dump	子模型导出按钮。

步骤3 配置好参数后，单击“Show Result”，解析原始网络模型。

如果模型转换后导出的.om模型文件中的原模型“Model Name”取值与Caffe原始网络模型文件（*.prototxt）或者TensorFlow原始网络模型文件（*.pb）中的“Model Name”取值不同时，解析网络模型时会弹出如图9-21所示界面。

- 单击“Yes”继续解析，进入图9-22。
- 单击“No”，返回子模型导出界面。

图 9-21 Model Name 取值不同

解析成功的原始网络模型如图9-22所示。

图 9-22 解析成功结果

其中：

- 红色节点表示进行Vector算法精度比对时，余弦相似度结果小于阈值的算子。
- 若某个节点包含多个算子，单击该节点后，该节点会出现绿色选中框，图9-22中的中间部分会展示该节点所包含的所有算子。
  - **查看算子信息**

    单击图9-22中间区域框上方的 ^ 展开算子的所有参数信息，用户也可以逐层单击相关参数左侧的展开按钮 > ，查看参数的详细信息；单击 ˇ 折叠所有的算子信息。
  - **查看算子输出shape信息**

    图9-22还展示了每一层算子输出的维度和shape信息，如每一层算子连接线中间的NCHW，1,3,224,224等信息。
  - **搜索算子**

    在图9-22所示界面，单击中间区域的“Op Search”，在下方会弹出搜索区域。

    搜索区域中给出了该模型所用到的所有算子，您可以在搜索区域对话框中输入算子名称，下方搜索区域会列出相关的算子。选择其中一个算子，左侧网络拓扑结构中相应算子会显示绿色选中框，中间区域框上方会展示该层算子的详细信息，包括算子名称、算子输入、输出等信息。
  - **搜索算子内部信息**

    选中某层算子，单击图9-22中间区域框上方的 ^ 展开按钮，下方区域框会展示该层所包括所有算子的信息，然后单击 或在中间区域使用“Ctrl+F”快捷键，弹出图9-23所示搜索界面，其中：

- Case sensitive：大小写敏感。
- Wrap Search：循环搜索。
- Backward：反向搜索。

用户在输入框中输入想要查询的信息，比如算子的输入输出、属性等信息，然后在图9-23界面中勾选搜索特性，单击“Find”进行查找，如果匹配到相关信息，则中间区域框中相关信息会高亮显示，否则会提示“Value not found”。单击Close退出搜索。

图 9-23 搜索算子内部信息

步骤4 选中怀疑有问题的局部网络模型第一个节点（输入节点），右击选择“Start Layer”，该节点变成黄色；选中最后一个节点（输出节点）右击选择“End Layer”，该节点变成紫色，“Start Layer”与“End Layer”中间的节点出现绿色选中框，表示选中的局部网络，如图9-22选中的节点所示。

说明

当开始节点和结束节点选择不合法时，两个节点外框会变红，出现相应信息提示不合法原因，同时“Dump”按钮置灰，不可导出dump。可根据提示信息对开始节点和结束节点重新选择。

- 选中了输入和输出节点后，如果想更换输入或输出节点，则在新的节点上右击选择相应的属性即可，新的节点颜色随之变化，原有节点恢复原始颜色。
- 如果输入节点和输出节点为同一个，在当前节点右击选择“Start Layer”或“End Layer”，节点颜色随之变成黄色或紫色，同时弹出图9-24所示界面，单击“OK”，然后执行步骤5即可导出单个节点的子模型。

图 9-24 输入输出共节点

步骤5 确定输入节点和输出节点后，单击“Dump”导出子模型。

MindStudio后台会在原始网络模型中寻找子模型，然后将子模型导出。界面右下角会有导出进度提示，单击该进度条可以查看详细进度。

- 若出现图9-25所示信息，则说明导出成功。

图 9-25 导出成功提示

- 如果子模型导出时，找不到子模型某个算子输入层的shape信息时，会提示图9-26所示信息。该场景下，需要用户根据提示信息，查看原始模型后，自行输入相应算子输入层的shape信息，shape取值以英文冒号分隔。

  例如图9-26所示，提示res2a_branch1算子的输入pool1层算子的shape信息无法获取，查看原始模型后，pool1层算子的输出shape为“1:64:56:56”，则此处填写的shape信息为“1:64:56:56”（具体取值请根据实际情况进行修改）。

  图 9-26 找不到子模型某个算子输入层的 shape 信息

步骤6 导出成功后：

1. 进入MindStudio安装服务器“Dump Output Path”所在目录可以看到以时间戳命名的文件夹dump_result_*xxxx_xx_xx_xx_xx_xx*，其中：
   - dump.prototxt：导出的Caffe子模型文件。
   - dump.pb：导出的TensorFlow子模型文件。
   - dump.caffemodel：导出的子模型权重文件（Caffe模型特有）。
   - dump_data：选中的“Start Layer”与“End Layer”中间，子模型所包括的所有算子的dump数据信息。

     该文件夹中不仅包括导出子模型所包括的所有算子的dump信息，还包括子模型“Start Layer”层的输入数据，用于子模型推理时作为输入数据使用。

   用户可以将导出的子模型再次进行精度比对或发给华为工程师定位，如果问题解决则进入下一步验证。
2. MindStudio界面“Output”窗口可以看到导出日志（如下日志中的所有数据都为样例数据，请以实际导出模型为准）：

```
//导出过程中会为子模型添加输入信息
2020-05-22 16:27:57  Add Layer Into Prototxt:
name: "pool1_input_0"
type: "Input"
top: "conv1"
input_param {
  shape {
    dim: 1
    dim: 64
    dim: 112
    dim: 112
  }
}

//导出子模型的所有算子的dump数据信息
2020-05-22 16:27:57  Output Dump Path:/home/hisiso/dumpdate/dumpresult/ir6/
dump_result_2020_05_22_16_27_44/dump_data

//Start Layer与End Layer中间某层算子的dump信息，其中Output File表示Dump Layer层算子对应离线
```

```
模型中相应算子的输出dump数据，Input File表示Dump Layer层算子对应的离线模型中相应算子的输入dump数据
2020-05-22 16:27:58  Dump Layer:[res2a_branch2c, bn2a_branch2c, scale2a_branch2c]
Output File:Conv2D.res2a_branch2cres2ares2a_relu.151.1585884515795667
Input File:Conv2D.res2a_branch2bres2a_branch2b_relu.149.1585884515771632
...

//Start Layer与End Layer中间所有算子的dump信息
2020-05-22 16:27:58  Export
Layer:pool1,res2a_branch1,bn2a_branch1,scale2a_branch1,res2a_branch2a,bn2a_branch2a,scale2a_branch2a,res2a_branch2a_relu,res2a_branch2b,bn2a_branch2b,scale2a_branch2b,res2a_branch2b_relu,res2a_branch2c,bn2a_branch2c,scale2a_branch2c,res2a,res2a_relu
```

步骤7 导出子模型重新执行推理。

1. 将**步骤6.1**导出的子模型，参见**4 模型转换**转换成.om离线模型文件。

   如果导出的子模型不包括输入的data层节点，则转换.om离线模型时，请关闭AIPP预处理参数。

2. 将**步骤6.1**导出的子模型Start Layer层的输入数据，转换成bin格式，用户执行推理时，作为离线模型的输入数据使用。

   a. 查找Start Layer层的输入数据。

      输入数据为子模型导出时“Dump Layer”为“Start Layer”层算子对应的Input File文件，根据此信息，在MindStudio安装服务器“dump_data”目录下查找同名Input File文件，即为Start Layer层的输入数据。

   b. pb格式输入数据转成numpy格式

      进入MindStudio安装服务器，切换到***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux*/toolkit/tools/operator_cmp/compare**路径，执行如下命令将pb格式的输入数据转换成numpy格式：

      ```
      python3.7.5 shape_conversion.pyc -i pb格式输入数据绝对路径 -format NCHW -o 转换为numpy格式数据的存放路径
      ```

      上述命令中的路径请根据实际情况进行替换，转换完成后，可以在numpy格式数据的存放路径看到.npy格式的输入数据。

   c. numpy格式输入数据转成bin格式。

      将如下脚本中的数据路径替换为实际路径，然后将脚本另存为.py格式，例如submodeldataprocess.py。

      ```
      import numpy as np

      # 将numpy二进制格式转换为原始二进制格式
      conv1_relu_0 = np.load(
          "/home/hisisoc/dumpdate/subModelData/xxx.npy")  # .npy格式的输入数据的绝对路径
      conv1_relu_0 = conv1_relu_0.astype(np.float16)
      conv1_relu_0.tofile(
          "/home/hisisoc/dumpdate/subModelData/xxx.bin")  # 转换为bin格式数据的存放路径
      ```

      将submodeldataprocess.py脚本上传到MindStudio安装服务器任意路径，然后执行如下命令将.npy格式的输入数据转换成bin格式：

      ```
      python3.7.5 submodeldataprocess.py
      ```

      进入bin格式数据的存放路径，可以看到.npy格式的输入数据已经转换成bin格式。

3. 将导出的原始网络子模型在原始环境中执行推理业务，然后将**步骤7.1**和**步骤7.2**中的离线模型和输入数据在海思SoC板端环境执行推理业务，查看两者推理结果是否有差异。如果仍有差异，则参见**9 精度比对**使用**步骤6.1**中导出的原始网络子模型与**步骤7.1**原始网络子模型转换后的离线模型重新和原始模型标杆数据进行精度比对。

4. 重复执行子模型导出与子模型精度比对，把问题定位到最小模型，将导出后的数据发给华为工程师定位。

----结束

9.4 FAQ

9.4.1 Vector 比对报错：The data in dump file have zero.

故障现象

执行Vector比对时，界面弹出报错窗口，如图9-27所示。

图 9-27 Vector 比对报错信息：dump 数据文件包含 0

故障原因

导致该故障的可能原因是“Ouput”栏显示的dump数据文件中包含0，导致无法计算。

处理方法

删除“Output”栏显示的dump数据文件后，再执行Vector精度比对。

9.4.2 Vector 比对报错：ModuleNotFoundError: No module named 'google'

故障现象

执行Vector比对时，界面弹出报错窗口，如图9-28所示。

图 9-28 Vector 比对报错信息：未安装 protobuf

output栏的打印信息如下：

```
2020-01-21 09:38:17 Error: Traceback (most recent call last): File "/home/phisik3/vb_workspace/work_code/toolchain/operator_cmp/compare/CompareVector.py", line 17, in <module> File "/home/phisik3/vb_workspace/work_code/toolchain/operator_cmp/compare/AccumulatedRelativeError.py", line 3, in <module> File "/home/phisik3/vb_workspace/work_code/toolchain/operator_cmp/compare/VectorComparisonCommon.py", line 4, in <module> File "/home/phisik3/vb_workspace/work_code/toolchain/operator_cmp/net_inference/caffe/DumpData_pb2.py", line 6, in <module> ModuleNotFoundError: No module named 'google'
```

故障原因

操作系统未安装protobuf导致解析Caffe模型的dump数据出错。

处理方法

针对该问题现象，可以先使用下面命令安装protobuf后再进行精度比对：

使用MindStudio安装用户登录MindStudio服务器，执行命令**pip3.7.5 install protobuf --user**。

9.4.3 如何进行 dump 数据文件与 numpy 文件格式互转

数据转换脚本说明

本版本提供dump_data_conversion.pyc脚本，实现dump数据文件与numpy文件格式互转功能，具体命令行格式如下：

```
python3.7.5 dump_data_conversion.pyc -type TYPE -target TARGET -i INPUT_PATH -o OUTPUT_PATH
```

- -type：数据类型，必选参数 。参数值选项：
  - quant：量化Caffe模型数据。
  - tf：非量化TensorFlow模型数据。
  - caffe：非量化Caffe模型数据。
  - offline：离线模型数据。
- -target：数据转换目标格式，参考值选项：numpy和dump。必选参数 。
  - numpy：dump文件转换为numpy文件。
  - dump：numpy文件转换为dump文件。
- -i：数据文件路径（可以是文件夹或者文件），必选参数 。
  - numpy文件转dump文件：

    如果“i”参数输入是文件夹，则格式要求如下：文件夹下的文件必须是op_name.output_index.timestamp.npy 形式文件。

    如果“i”输入参数是文件名，则输入的文件名需满足“op_name.output_index.timestamp.npy ”格式要求。每次仅支持输入1个文件。

    其中op_name对应的名称需满足“A-Za-z0-9_-”正则表达式规则，output_index为0~9数字组成，timestamp为16位时间戳。
  - dump文件转numpy文件：

    如果“i”参数输入是文件夹，文件夹下的文件格式必须满足**9.2.1 比对数据格式要求**章节要求；如果“i”输入参数是文件名，则输入的文件名也必须满足**9.2.1 比对数据格式要求**章节要求。
- -o：转换后输出文件路径。必选参数。

说明

dump_data_conversion.pyc脚本存放位置：***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}-linux*/toolkit/tools/operator_cmp/compare**。

该脚本执行需依赖numpy、protobuf、scipy库，需要执行命令**pip install numpy protobuf scipy --user**进行安装。如果安装MindStudio时已安装，则不需要执行。

使用该脚本进行数据转换，确保主机内存大小不低于15GB；如果待转换的dump数据单个文件大小超过441MB，则建议使用更大内存主机。

转换 Caffe 原始数据 npy 文件为 dump 数据文件

获取到Caffe原始数据后，将npy文件传到MindStudio服务器上，通过dump_data_conversion.pyc脚本进行数据转换，输出二进制格式的dump文件。

Caffe原始数据npy文件转dump命令举例：

python3.7.5 dump_data_conversion.pyc -type caffe -target dump -i */home/HwHiAiUser/caffenpyfile* -o */home/HwHiAiUser/caffedump*

说明

- /home/HwHiAiUser/caffenpyfile：npy文件所在路径。
- /home/HwHiAiUser/caffedump：生成的dump文件存放路径。

转换 TensorFlow 原始数据 npy 文件为 dump 数据文件

获取到原始数据后，将npy文件传到MindStudio服务器上，通过dump_data_conversion.pyc脚本进行数据转换，输出二进制格式的dump文件。

TensorFlow原始数据npy文件转dump命令举例：

python3.7 dump_data_conversion.pyc -type tf -target dump -i */home/HwHiAiUser/tfnpyfile* -o */home/HwHiAiUser/tfdump*

说明

- /home/HwHiAiUser/tfnpyfile：npy文件所在路径。
- /home/HwHiAiUser/tfdump：生成的dump文件存放路径。

9.4.4 如何查看 dump 数据文件

dump文件无法通过文本工具直接查看其内容，为了查看dump文件内容，我们提供以下脚本将dump文件转换为numpy格式文件后，再通过numpy官方提供的能力转为txt文档进行查看：

步骤1 使用安装用户登录开发环境。

步骤2 进入/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare目录。

步骤3 执行dump_data_conversion.pyc脚本，转换dump文件为numpy文件。举例：

python3.7.5 dump_data_conversion.pyc -target numpy -type offline -i */home/HwHiAiUser/dump* -o */home/HwHiAiUser/dumptonumpy*

说明

dump_data_conversion.pyc脚本的各个输入参数使用方法，请参考**9.4.3 如何进行dump数据文件与numpy文件格式互转**

步骤4 调用Python，转换numpy文件为txt文件。

$ python3.7.5

```
Python 3.7.5 (default, Mar  5 2020, 16:07:54)[GCC 5.4.0 20160609] on linuxType "help", "copyright", "credits" or "license" for more information.
```

>>> import numpy as np

>>> a = np.load("/home/HwHiAiUser/dumptonumpy/*Pooling.pool1.1147.1589195081588018.output.0.npy*")

>>> b = a.flatten()

>>> np.savetxt("/home/HwHiAiUser/dumptonumpy/*Pooling.pool1.1147.1589195081588018.output.0.txt*", b)

转换为.txt格式文件后，维度信息、Dtype均不存在。详细的使用方法请参考numpy官网介绍。

----结束

9.4.5 如何批量处理生成的 npy 文件名异常情况

TensorFlow模型生成dump数据时，因tfdbg自身原因或运行环境原因，会出现tfdbg截断算子名，导致生成的npy文件名与预期不符，造成转换dump数据文件异常。此时需要参考下面方法重新生成npy文件，使得npy文件名符合精度比对要求。

说明

- 本文中脚本名称、路径等均为举例，请根据实际替换。
- 批量处理后，如果遇到某算子的dump文件存在，但是比对结果为NaN，需要检查该算子的dump文件名中的{op_name}是否与TensorFlow算子名称一致，如果不一致需要手动修改dump文件名中的算子字段与TensorFlow算子名称一致。其中如果出现"/"请修改为"_"。

步骤1 执行TensorFlow工程。

进入调试命令行交互模式后，输入run命令。

步骤2 执行**lt > tensor_name**命令将所有tensor的名称暂存到文件里。

步骤3 创建可执行脚本，如pt_cmd.sh，获取**tensor_name**文件中tensor_name对应的tensor_index。

脚本内容如下：

```
#!/bin/bash
timestamp=$[$(date +%s%N)/1000]
index=1
while read -r line
do
  tensor_index=`echo $line | awk '{print $4}'`
  echo "pt "$tensor_index" -n 0 -w "$((index++))"."$timestamp".npy" >> $2
done < $1
```

赋予pt_cmd.sh可执行权限并执行脚本。

chmod +x *pt_cmd.sh*

bash *pt_cmd.sh tensor_name tensor_name.txt*

步骤4 回到tfdbg命令行，输入run命令后，将上一步生成的***tensor_name.txt***文件内容粘贴执行，生成npy文件。

步骤5 将生成的npy文件，移动到新的文件夹，如npy_dir。

步骤6 创建可执行脚本，如index_to_tensorname.sh，并执行脚本批量修改npy文件名。

脚本内容如下：

```
#!/bin/bash
timestamp=$[$(date +%s%N)/1000]
while read -r line
do
  tensor_index=`echo $line | awk '{print $2}'`
  real_file=`echo $line | awk '{print $6}'`
  changed1_tensor_index=${tensor_index//\//_}
  changed2_tensor_index=${changed1_tensor_index//:/.}
  echo $2/$real_file $2/$changed2_tensor_index"."$timestamp".npy"
  if [ -r $2/$real_file ]
  then
    mv $2/$real_file $2/$changed2_tensor_index"."$timestamp".npy"
  fi
done < $1
```

赋予index_to_tensorname.sh可执行权限并执行脚本。

chmod +x *index_to_tensorname.sh*

bash *index_to_tensorname.sh tensor_name.txt npy_dir*

----结束

10 Profiling 性能分析

10.1 应用工程Profiling
10.2 导入并展示Profiling结果

10.1 应用工程 Profiling

10.1.1 简介与约束

Profiling 简介

MindStudio针对昇腾AI处理器提供高效、易用的系统化性能分析工具Profiling，便于快速识别产品的关键性能瓶颈并提出针对性能优化的建议，最终实现产品的极致性能。

Profiling提供针对APP工程的硬件和软件性能数据采集、分析、汇总展示。总体流程如下：

1. 运行Profiling采集。

   确保APP工程可正常执行的条件下，用户在配置界面开启Profiling开关。

2. Profiling采集性能数据。

   MindStudio编译当前工程生成可执行文件，并将可执行文件拷贝到设备侧，MindStudio向Profiling工具下发数据采集指令，由Profiling工具完成设备侧数据采集任务，采集结束后，将生成的数据文件拷贝到MindStudio侧。

3. MindStudio查询并解析数据。

   Profiling采集结束后，MindStudio调用Profiling工具接口查询数据，并将数据以json格式存储在“工程目录~/profiling ”目录下。

4. MindStudio展示性能数据。

   MindStudio通过对json文件做数据处理，生成前端展示视图数据，此时用户可通过右键单击工程文件名并选择“View Profiling Result”菜单进行数据图形化展示。

Profiling 使用约束

以下情况不能使用Profiling功能：

- Profiling不支持同时从MindStudio安装侧发起2个基于相同结果目录的Profiling。
- Profiling功能与Dump功能不建议同时使用，即启动Profiling前，请关闭数据Dump。原因：如果同时开启，由于Dump操作会影响系统性能，会造成Profiling采集的性能数据指标不准确。Dump相关介绍请参考**9.2.2.1 准备离线模型dump数据**。
- Profiling工具需要配套python3.7版本使用，推荐使用python3.7.5版本。

10.1.2 Profiling 前准备

请参见**7 应用开发**完成工程文件开发，并通过MindStudio编译、运行，确保APP工程正常运行。

10.1.3 启动 Profiling

通过以下操作方法启动Profiling：

步骤1 选择已编译完成的工程。

步骤2 单击菜单栏“Run > Run...”，在弹出窗口中选择“Edit Configurations...”，弹出Profiling设置窗口 。如**图10-1**所示。

图 10-1 Profiling 设置界面

表 10-1 Profiling 参数说明

参数	说明
Name	运行配置名称，可修改。
Executable	选择可执行文件的位置。
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。 windows使用场景下没有本项。

版权所有 © 华为技术有限公司

参数		说明
SSH Connection		SSH连接，选择Remote Run模式时可见，配置与修改具体请参见13.1 SSH连接管理。 下拉选择远程运行应用的服务器的IP地址，若未添加，请单击进行添加，详情请参见13.1.2 新增SSH连接。
Command Arguments		运行应用传入的参数，和用户的代码实现有关，为可选配置。
Environment Variables		环境变量配置。可以直接在框中输入也可以单击后在出的弹窗内单击填写。 **说明** 当前还不支持配置PATH路径的环境变量。
Toolkit Path		远端运行环境toolkit软件包安装路径。选择Remote Run模式时可见，一般为可选配置，Profiling性能分析时为必选项。例如配置为/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit。
Enable Profiling		是否开启Profiling相关功能，详细配置。
Collect AI Core Trace	Task-Based	AI Core采集开关，以task为粒度进行性能数据采集。 ● Pipe Utilization：采集计算单元和搬运单元耗时占比。 ● UB L1 L2 Main Memory Bandwidth：采集内存读写速率。
	Sample-Based	AI Core采集开关，以固定的时间周期（AI Core-Sampling Interval）进行性能数据采集。 ● Pipe Utilization和UB L1 L2 Main Memory Bandwidth，含义同上。 ● Frequency(hz)：采集频率，默认值为10，单位hz。
Collect AI Stack	Ascend Compute Language (ACL)	ACL采集开关，采集接口流水信息。
	Graph Engine (GE)	Graph Engine采集开关，采集模型图引擎调度流水信息。
	Runtime	Runtime采集开关，采集运行管理器接口流水信息。
	System Task	System Task采集开关，采集系统类任务调度流水信息。
Before launch		此栏中可配置运行前的预处理操作，为可选配置。 例如，单击添加“Ascend APP Build”，在运行可执行文件前，编译可执行文件。

版权所有 © 华为技术有限公司

注意

如果未在Profiling设置窗口中配置Toolkit Path，则需要在应用工程所在的远端运行环境中设置环境变量，操作如下：

1. Ascend EP环境下，以安装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以下内容。
   ```
   export PATH={install_path}/toolkit/tools/profiler/bin:$PATH
   ```
   其中{install_path}为Ascend-cann-toolkit包安装路径，例如：/home/HwHiAiUser/Ascend/ascend-toolkit/latest。
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。

步骤3 单击“Run”，执行工程的Profiling。

工程执行完成后，MindStudio窗口下方展示Profiling结果视图。

说明

Profiling的过程日志信息可以在下方任务栏的“Run”窗口中查看。

----结束

10.1.4 查看 Profiling 数据

Profiling执行成功后，通过右键单击工程目录，选择“View Profiling Result”菜单，查看Profiling数据结果。

图 10-2 查看 Profiling 数据

Profiling结果展示视图位于MindStudio窗口下方，如图10-3所示。

图 10-3 Profiling 视图窗口

版权所有 © 华为技术有限公司

因Profiling支持Task-Based和Sample-Based两种不同方式的AI Core采集方式，以及两种不同的Metrics采集项（Pipe Utilization和UB L1 L2 && Main Memory Bandwidth），因此，会出现Profiling各页签展示结果不同，本文不再分别截图区分。

表 10-2 AI Core 采集方式视图差异

视图	Task-Based	Sample-Based
Summary View（Project info & AI Core Op Statistic）	展示	展示
Summary View（AI Core Utilization）	不展示	展示
Timeline	展示	展示
AI Core Op Detail	展示	不展示
AI Core Metrics	不展示	展示
Runtime API	展示	展示

Timeline 视图

在Profiling运行完成后，以时序图的呈现方式为用户提供全流程推理过程中的运行情况，按照调度流程ACL> GE > Runtime > Ascend Compute来呈现整体的运行状况。

说明

- “Start Time”和“End Time”表示本次性能采集数据的展示时间范围，图中数据展示的时间范围可通过调整右上方的滑块进行变更，也可以通过“Ctrl+鼠标滑轮滚动”进行变更。
- 鼠标放到对应的时间块上，可以查看具体时间段信息，“Current Time”表示鼠标所在时间块的开始时间。
- 拖动页面右上方时间进度条，界面会重新请求数据，部分模块在该时间段因为没有相关数据而不展示出来。

- ACL：展示模型、算子、Runtime API等耗时数据。如果没有某一类数据则不显示。

图 10-4 ACL

- GE：展示模型的数据输入、推理、数据输出耗时。

图 10-5 GE

版权所有 © 华为技术有限公司

- Runtime：展示每个线程调用runtime API的时序信息，如图10-6所示。

说明

鼠标放到对应的时间块上除了展示时间段信息外，compute视图还展示task运行状态pending、waiting、running；stream、scheduling视图展示task的running运行状态。

图 10-6 Thread 分析

- Ascend Compute：展示每个Stream的System task时序信息，如图10-7所示。

图 10-7 Streams 分析

Runtime API 视图

Runtime性能数据展示，显示Runtime的API调用信息。

Runtime API指标统计各类Runtime API的调用次数以及API的调用时长，过多的调用次数以及调用时长表示算子开发中算子的调度逻辑可能存在问题。

说明

图10-8中Stream ID列显示的“N/A”表示该API是直接调用的接口，不属于任何Stream。

版权所有 © 华为技术有限公司

图 10-8 Runtime API 调用信息

Name	Stream ID	Time(%)	Time(ns)	Calls	Avg(ns)	Min(ns)	Max(ns)
DevBinaryRegister	N/A	0.201844	331849	60	5530	3112	18891
DevBinaryUnRegister	N/A	0.153726	252739	60	4212	2429	10112
FunctionRegister	N/A	0.122158	200839	60	3347	2574	8816
KernelLaunch	517	7.621503	12530417	60	208840	41970	9468752
KernelFusionStart	517	0.010895	17913	1	17913	17913	17913
KernelFusionEnd	517	0.005174	8506	1	8506	8506	8506
StreamCreate	517	0.293582	482674	3	160891	33114	245535
StreamDestroy	517	0.030658	50405	3	16801	5171	32877
DevMalloc	N/A	0.391161	643103	65	9893	4714	127534
DevFree	N/A	2.533971	4166069	65	64093	1796	3541285
HostMalloc	N/A	0.09334	153459	4	38364	15468	73076
HostFree	N/A	0.384407	631999	4	157999	10285	388407
ManagedMemAlloc	N/A	0.051794	85154	60	1419	980	13296
ManagedMemFree	N/A	0.03256	53531	60	892	850	2303
MemCopy	N/A	16.40447	26970382	189	142700	10383	24074744
DeviceReset	N/A	58.148344	95600957	1	95600957	95600957	95600957
ContextCreate	N/A	4.878983	8021475	1	8021475	8021475	8021475
ContextDestroy	N/A	4.799002	7889978	1	7889978	7889978	7889978

表 10-3 Runtime API 页面部分参数说明

参数名	参数含义
Name	调用的API名称。
Stream ID	API运行的stream ID。
Time(%)	总耗时占比。
Time(ns)	总耗时。
Calls	API被调用的次数。
Avg(ns)、Min(ns)、Max(ns)	分别对应平均耗时、最短耗时、最长耗时。

AI Core Op Detail 视图

展示推理过程中AI Core算子的执行详细内容，包括HWTS、GE Data、PMU Data三部分。

采集项选择“Pipe Utilization”时采集结果如**图10-9**所示。

该指标统计整个Profiling采样周期中各个AI Core类型指令在所有类型指令中的耗时占比，其中Mac Ratio，Vec Ratio，Scalar Ratio分别代表cube指令利用率、vec指令利用率、scalar指令利用率，是需要重点关注的指标。

版权所有 © 华为技术有限公司

图 10-9 “Pipe Utilization”采集结果信息

AI Core Op Detail　Device ID: 0　Op Type All

☑ GE Data ☑ PMU Data

Op Name	Task ID	OP Type	Task Type	Stream ID	Total Cycles	Aicore Time(us)	Task Start Time	Task Duration(us)	Task Wait Time(us)	Model Name	Block Dim	Input Shapes
trans_Cast_169	62	Cast	AI CORE	517	3021	2.221324	175129836421033	20.571	16.144	resnet50	2	"1,1000"
prob	61	SoftmaxV2	AI CORE	517	2215	3.257353	175129836384318	20.571	20.832	resnet50	1	"1,1000"
trans_TransData_1...	60	TransData	AI CORE	517	1823	2.680882	175129836330833	32.653	0.0	resnet50	1	"63,1,16,16"
fc1000	59	FullyConnection	AI CORE	517	132577	97.483088	175129836234273	115.568	0.0	resnet50	2	"1,128,1,1,16";"12...
pool5	58	Pooling	AI CORE	517	29509	21.697794	175129836204534	78.019	17.916	resnet50	2	"1,128,7,7,16"
res5c_branch2cres...	57	Conv2D	AI CORE	517	67223	49.428676	175129836120893	65.725	0.0	resnet50	2	"1,32,7,7,16";"32,1...
res5c_branch2bres...	56	Conv2D	AI CORE	517	136642	100.472059	175129836004905	167.601	0.0	resnet50	2	"1,32,7,7,16";"288...
res5c_branch2ares...	55	Conv2D	AI CORE	517	61687	45.358088	175129835946261	123.643	15.936	resnet50	2	"1,128,7,7,16";"12...
res5b_branch2cres...	54	Conv2D	AI CORE	517	65560	48.205882	175129835842359	87.966	0.0	resnet50	2	"1,32,7,7,16";"32,1...
res5b_branch2bres...	53	Conv2D	AI CORE	517	133349	98.050735	175129835764912	152.341	19.113	resnet50	2	"1,32,7,7,16";"288...
res5b_branch2ares...	52	Conv2D	AI CORE	517	61584	45.282353	175129835671533	74.266	0.0	resnet50	2	"1,128,7,7,16";"12...
res5a_branch1res5...	51	Conv2D	AI CORE	517	125602	92.354412	175129835588617	144.112	18.645	resnet50	2	"1,64,7,7,16";"64,1...
res5a_branch2c	50	Conv2D	AI CORE	517	62244	45.767647	175129835510549	59.423	0.0	resnet50	2	"1,32,7,7,16";"32,1...
res5a_branch2bres...	49	Conv2D	AI CORE	517	135193	99.406618	175129835405081	152.393	17.864	resnet50	2	"1,32,7,7,16";"288...
res5a_branch2ares...	48	Conv2D	AI CORE	517	32811	24.125735	175129835342168	45.049	0.0	resnet50	2	"1,64,7,7,16";"64,3...
res4f_read_selectr...	47	ReadSelect	AI CORE	517	24860	18.279412	175129835313471	61.873	18.28	resnet50	2	"1,64,14,14,16";"1...
res4f_branch2bres...	46	Conv2D	AI CORE	517	36401	26.765441	175129835252383	42.808	0.0	resnet50	2	"1,16,14,14,16";"1...
res4f_branch2ares...	45	Conv2D	AI CORE	517	16928	24.894118	175129835201917	80.412	0.0	resnet50	1	"1,64,14,14,16";"6...
res4e_branch2cres...	44	Conv2D	AI CORE	517	35278	25.939706	175129835169835	50.414	20.363	resnet50	2	"1,16,14,14,16";"1...
res4e_branch2bres...	43	Conv2D	AI CORE	517	39296	28.894118	175129835096612	52.86	0.0	resnet50	2	"1,16,14,14,16";"1...
res4e_branch2ares...	42	Conv2D	AI CORE	517	16987	24.980882	175129835073592	60.726	18.801	resnet50	1	"1,64,14,14,16";"6...
res4d_branch2cres...	41	Conv2D	AI CORE	517	36261	26.6625	175129835020836	33.955	0.0	resnet50	2	"1,16,14,14,16";"1...
res4d_branch2bres...	40	Conv2D	AI CORE	517	40339	29.661029	175129834974118	65.675	18.331	resnet50	2	"1,16,14,14,16";"1...

采集的事件详细描述如下：

- HWTS采集数据：
  - Op Name：算子名称。
  - Task ID：Task对应的ID。
  - OP Type：算子类型。
  - Task Type：Task类型。
  - Stream ID：对应的Stream ID。
  - Total Cycles：该Task的所有指令的cycle总数。
  - Task Start Time：Task开始时间。
  - Task Duration：Task耗时。
  - Task Wait Time：上一个Task的结束时间与当前Task的开始时间间隔。
- PMU Data采集数据：
  - Mac Ratio：代表cube类型指令（矩阵类运算指令）的cycle数在所有指令的cycle数中的占用比。
  - Mac Time：代表cube类型指令（矩阵类运算指令）耗时。
  - Vec Ratio：代表vec类型指令（向量类运算指令）的cycle数在所有指令的cycle数中的占用比。
  - Vec Time：代表vec类型指令（向量类运算指令）耗时。
  - Scalar Ratio：代表scalar类型指令（标量类运算指令）的cycle数在所有指令的cycle数中的占用比。
  - Scalar Time：代表scalar类型指令（标量类运算指令）耗时。
  - Mte1 Ratio：代表mte1类型指令（L1->L0A/L0B搬运类指令）的cycle数在所有指令的cycle数中的占用比。
  - Mte1 Time：代表mte1类型指令（L1->L0A/L0B搬运类指令）耗时。
  - Mte2 Ratio：代表mte2类型指令（DDR->AI Core搬运类指令）的cycle数在所有指令的cycle数中的占用比。
  - Mte2 Time：代表mte2类型指令（DDR->AI Core搬运类指令）耗时。
  - Mte3 Ratio：代表mte3类型指令（AI Core->DDR搬运类指令）的cycle数在所有指令的cycle数中的占用比。

  - Mte3 Time：代表mte3类型指令（AI Core->DDR搬运类指令）耗时。
  - Icache Miss Rate：代表icache缺失率，即未命中icache，数值越小越好。
  - Memory Bound：用于识别AI Core执行算子计算过程是否存在Memory瓶颈，由Mte2 Ratio/max(Mac Ratio, Vec Ratio)计算得出。计算结果小于1，表示没有Memory瓶颈；计算结果大于1 则表示有Memory瓶颈，且数值越大越瓶颈严重。
- GE Data采集数据：
  - Block Dim：Task运行时所在的核。
  - Input Shapes：算子的输入维度。
  - Input Data Types：input算子输入数据类型。
  - Input Formats：input算子格式。
  - Output Shapes：算子的输出维度。
  - Output Data Types：output算子输出数据类型。
  - Output Formats：output算子格式。

采集项选择“UB L1 L2 && Main Memory Bandwidth”时采集结果如**图10-10**所示。

该指标统计整个Profiling采样周期中各个内存读写速率。

图 10-10 “UB L1 L2 && Main Memory Bandwidth”采集结果信息

AI Core Op Detail　Device ID: 0　Op Type All

☐ GE Data ☑ PMU Data

Op Name	Task ID	...	Task Duration(us)	Task Wait Time(us)	Ub Read Bw(GB/s)	Ub Write Bw(GB/s)	L1 Read Bw(GB/s)	L1 Write Bw(GB/s)	L2 Read Bw(GB/s)	L2 Write Bw(GB/s)	Main Mem Read Bw(GB/s)	Main Mem Write Bw(
trans_TransData_0	3	...	72.082	0.0	142.036643	17.727694	0.0	0.0	0.0	35.50067	6.80354	0.0
conv1conv1_relu	4	...	172.184	0.0	37.810972	18.72837	280.389085	36.908359	17.118993	10.681489	1.485054	0.0
pool1	5	...	115.362	0.0	113.948855	121.590215	0.0	0.0	88.358121	20.446507	0.547674	0.0
res2a_branch2ares2a_bra...	6	...	53.278	24.373	107.825286	55.374085	48.212517	65.569024	31.498845	31.498845	2.711954	0.0
res2a_branch2bres2a_bra...	7	...	65.153	0.0	44.20919	21.204009	139.065801	44.02524	17.597833	12.018032	5.173579	0.0
res2a_branch2c	8	...	65.151	0.0	118.957212	55.593937	13.88299	27.76598	13.88299	55.531959	2.64733	0.0
res2a_branch1res2ares2a...	9	...	63.171	0.0	123.041459	105.473831	158.411856	22.573689	48.513631	38.810905	2.314298	0.0
res2b_branch2ares2b_bra...	10	...	64.63	0.0	64.597008	30.440809	89.036881	144.154951	69.250908	17.312727	3.466962	0.0
res2b_branch2bres2b_bra...	11	...	59.632	0.0	44.438925	21.317893	139.812703	44.261693	17.692348	12.082579	5.201365	0.0
res2b_branch2cres2bres2...	12	...	62.496	0.0	122.451399	104.700505	157.158189	22.395042	48.129696	38.503756	2.295983	0.0
res2c_branch2ares2c_bra...	13	...	46.246	0.0	64.383371	30.332357	88.719669	143.641369	69.004187	17.251047	3.45461	0.0
res2c_branch2bres2c_bra...	14	...	48.018	25.103	24.510357	12.889389	150.419726	63.719467	31.859734	7.31207	11.527702	0.0
res2c_read_selectres2c_b...	15	...	60.622	0.0	43.914014	60.711535	8.936799	8.179443	81.642956	9.896116	2.006993	0.0
res3a_branch2ares3a_bra...	16	...	34.475	0.0	53.332157	27.390245	101.521254	81.973683	30.897773	15.448886	11.389613	0.0
res3a_branch2bres3a_bra...	17	...	47.341	19.791	21.145972	11.381521	126.255474	44.99671	12.96964	6.48482	9.909202	0.0
res3a_branch2c	18	...	36.507	0.0	87.114846	34.488202	56.712716	39.55887	8.576923	34.307693	11.804209	0.0
res3a_branch1res3ares3a...	19	...	58.956	20.155	90.766875	62.099447	78.678284	50.229639	32.671491	21.780994	15.099452	0.0
res3b_branch2ares3b_bra...	20	...	45.673	0.0	30.369645	16.888366	38.4339	76.867801	38.4339	9.608475	12.991051	0.0
res3b_branch2bres3b_bra...	21	...	43.435	21.561	21.331859	11.501817	127.589926	45.472301	13.106722	6.553361	10.013937	0.0
res3b_branch2cres3bres3...	22	...	59.527	0.0	97.126042	85.807065	66.634712	41.25006	46.644298	31.096199	6.247004	0.0
res3c_branch2ares3c_bra...	23	...	36.559	0.0	31.19282	17.342629	39.467695	78.935391	39.467695	9.866924	13.340484	0.0
res3c_branch2bres3c_bra...	24	...	46.82	18.801	21.04802	11.35545	125.966273	44.89364	12.939932	6.469966	9.886504	0.0

采集的事件详细描述如下：

- HWTS采集数据：
  - Op Name：算子名称。
  - Task ID：Task对应的ID。
  - Stream ID：对应的Stream ID。
  - Total Cycles：该Task的所有指令的cycle总数。
  - Task Start Time：Task开始时间。
  - Task Duration：Task耗时。
  - Task Wait Time：Task等待时长。
- PMU Data采集数据：
  - Ub Read Bw：代表UB读带宽速率，单位GB/s。

  - Ub Write Bw：代表UB写带宽速率，单位GB/s。
  - L1 Read Bw：代表L1读带宽速率，单位GB/s。
  - L1 Write Bw：代表L1写带宽速率，单位GB/s。
  - L2 Read Bw：代表L2读带宽速率，单位GB/s。
  - L2 Write Bw：代表L2写带宽速率，单位GB/s。
  - Main Mem Read Bw：代表主存储器读带宽速率，单位GB/s。
  - Main Mem Write Bw：代表主存储器写带宽速率，单位GB/s。
- GE Data采集数据：
  - Block Dim：Task运行时所在的核。
  - Input Shapes：算子的输入维度。
  - Input Data Types：input算子输入数据类型。
  - Input Formats：input算子格式。
  - Output Shapes：算子的输出维度。
  - Output Data Types：output算子输出数据类型。
  - Output Formats：output算子格式。

Summary View 视图

支持Summary View视图展示，呈现APP工程信息、AI Core Op Statistic信息、算子Memory Bandwidth信息与AI Core Utilization信息。**图10-11**展示结果为Sample-Based场景下“Metrics”采集项设置为“UB L1 L2 && Main Memory Bandwidth”时展示结果。

图 10-11 Summary View 视图

Model Name	OP Type	Core Type	Count	Total Time(us)	Min Time(us)	Avg Time(us)	Max Time(us)	Ratio(%)
resnet50	Conv2D	AI Core	100	8590.932	46.349	85.909	188.795	83.337645
resnet50	Pooling	AI Core	4	428.155	75.671	107.038	135.151	4.153383
resnet50	ReadSelect	AI Core	6	425.383	48.746	70.897	82.599	4.126493
resnet50	TransData	AI Core	4	338.313	45.568	84.578	131.298	3.281857
resnet50	FullyConnection	AI Core	2	284.987	126.816	142.493	158.171	2.76456
resnet50	SoftmaxV2	AI Core	2	175.457	83.015	87.728	92.442	1.702047

- APP工程信息展示工程的基本信息。如**表10-4**所示。
- 算子Memory Bandwidth信息以折线图的形式呈现各个算子的Main Memory情况，仅在Task-Based场景下且AI Core采集项为“UB L1 L2 && Main Memory Bandwidth”时展示。横轴表示时间，纵轴表示Main Memory读写带宽；另外，可以通过“Inference Iteration Index”查看推理的不同迭代数据。
- AI Core Utilization数据用以呈现推理运行期间各Device上AI Core的利用率情况，仅在Sample-Based场景下呈现。横轴表示采集时长，纵轴表示利用率。

版权所有 © 华为技术有限公司

- 当采用Task-Based采集方式时，通过单击表格中“OP Type”列对应的算子类型，可以跳转到AI Core OP Detail视图，按照所选择的算子类型进行过滤，并可按照算子耗时排序查看对应算子类型的所有算子数据。
- AI Core Op Statistic信息展示算子的基本信息。如表10-5所示。

表 10-4 APP 工程信息说明

参数	参数说明	样例
App Name	工程名称。	MyApp
Duration	运行时长、起始时间。	2.004 s @ 2020-06-29 14:56:25
Target Host IP	Host侧IP地址。	10.175.*.*
Command Arguments	保留参数	-
Output Path	MindStudio侧json解析文件路径。	/home/user/Ascendproject/MyApp/profiling/
Target System	操作系统信息。	lhisilinux

表 10-5 Op 耗时性能分析说明

参数名	参数含义
Model Name	模型名称。
OP Type	算子类型。
Core Type	Core类型。
Count	算子调用次数。
Total Time(us)	算子调用总耗时。
Avg Time(us)、Min Time(us)、Max Time(us)	分别对应算子调用平均耗时、最小耗时、最大耗时。
Ratio(%)	该类算子在对应模型中的耗时占比。

10.2 导入并展示 Profiling 结果

MindStudio提供在线导入已有Profiling结果数据并进行解析展示的功能，具体操作方法如下：

单击菜单栏“Ascend > Import Profiling Result”，在弹出的窗口中选择已有Profiling结果数据所在目录，单击“Select”，页面展示Profiling数据解析结果。

说明

不支持单算子Profiling结果导入。

解析结果说明，请参考**10.1.4 查看Profiling数据**。

11 System Profiling 工具

11.1 功能介绍
11.2 使用前准备
11.3 使用指南

11.1 功能介绍

System Profiling实现了Host+Device侧丰富的性能数据采集能力和全景Timeline交互分析能力，展示Host+Device侧各项性能指标，帮助用户快速发现和定位AI应用的性能瓶颈。包括资源瓶颈导致的AI算法短板，指导算法性能提升和系统资源利用率的优化。System Profiling支持Host+Device侧的资源利用可视化统计分析，具体包括Host侧CPU、Memory、Disk、Network利用率、AI Core Metrics利用率和memory bandwidth（DDR等的带宽）等类型指标。

System Profiling当前只支持推理场景。

注意

- System Profiling工具仅适用于开发和测试环境，供开发者使用，禁止在生产环境使用。如果使用在生产环境，存在sudo提权风险。
- 使用dump功能会占用昇腾AI处理器能力，可能导致profiling结果偏差，不建议同时使用。

11.2 使用前准备

11.2.1 安装工具依赖

11.2.1.1 Ubuntu x86/ARM

说明

请使用MindStudio安装用户执行如下操作。

步骤1 执行如下命令，安装相关工具和依赖。

```
sudo apt-get install iotop
sudo apt-get install -y linux-tools-`uname -r`
sudo apt-get install lsof
sudo apt-get install net-tools
pip3.7.5 install django
pip3.7.5 install django-sslserver
pip3.7.5 install psutil
pip3.7.5 install APScheduler
pip3.7.5 install six
```

说明

非root用户在执行**pip**命令安装相关工具和依赖时，需要在后面加上**--user**，例如：**pip3.7.5 install django --user**。

步骤2 以安装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如下内容。

```
export PATH=/usr/sbin:$PATH
```

步骤3 执行**:wq!**命令保存文件并退出。

步骤4 执行**source ~/.bashrc**命令使其立即生效。

步骤5 执行**ltrace --version**命令查询ltrace工具版本，如果不是0.7.3则需要参考如下命令安装ltrace 0.7.3。

```
sudo apt-get install libelf-dev
wget https://ltrace.org/ltrace_0.7.3.orig.tar.bz2 --no-check-certificate
tar xjvf ltrace_0.7.3.orig.tar.bz2
cd ltrace_0.7.3
./configure --disable-werror
make
make install
```

说明

ARM架构操作系统请把wget命令替换为wget **http://ports.ubuntu.com/pool/main/l/ltrace/ltrace_0.7.3-6ubuntu1_arm64.deb**，需要安装dpkg后执行dpkg --unpack ltrace_0.7.3-6ubuntu1_arm64.deb解压。

----结束

11.2.1.2 Euler 2.8 ARM

说明

请使用MindStudio安装用户执行如下操作。

步骤1 执行如下命令，安装相关工具和依赖。

```
sudo yum install dpkg
sudo yum install lsof
sudo yum install perf
sudo yum install net-tools
pip3.7.5 install django
pip3.7.5 install django-sslserver
pip3.7.5 install psutil
pip3.7.5 install APScheduler
pip3.7.5 install six
```

说明

非root用户在执行**pip**命令安装相关工具和依赖时，需要在后面加上**--user**，例如：**pip3.7.5 install django --user**。

步骤2 以安装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如下内容。

```
export PATH=/usr/sbin:$PATH
```

步骤3 执行**:wq!**命令保存文件并退出。

步骤4 执行**source ~/.bashrc**命令使其立即生效。

步骤5 安装iotop 至/usr/local/python3.7.5/lib/python3.7/site-packages目录。

```
wget http://guichaz.free.fr/iotop/files/iotop-0.6.tar.bz2
tar -xvf iotop-0.6.tar.bz2
cd iotop-0.6
sed -i 's/itervalues/values/g' setup.py
python3.7.5 setup.py build
python3.7.5 setup.py install
ln -s /usr/local/python3.7.5/sbin/iotop /usr/sbin/iotop
```

步骤6 执行**ltrace --version**命令查询ltrace工具版本，如果不是0.7.3则需要参考如下命令安装ltrace 0.7.3。

```
sudo yum install libelf-dev
wget http://ports.ubuntu.com/pool/main/l/ltrace/ltrace_0.7.3-6ubuntu1_arm64.deb
dpkg --unpack ltrace_0.7.3-6ubuntu1_arm64.deb
cd ltrace_0.7.3
./configure --disable-werror
make
make install
```

----结束

11.2.2 使用前准备

硬件要求

安装MindStudio的开发环境必须配套昇腾设备。

环境准备

在昇腾设备上安装好驱动、固件和Ascend-cann-toolkit开发套件包。请参见**7 应用开发**完成工程文件开发，并通过MindStudio编译、运行，确保APP工程正常运行。

步骤1 root用户登录环境。

步骤2 打开“/etc/sudoers”文件：

```
chmod u+w /etc/sudoers
vi /etc/sudoers
```

步骤3 在该文件中增加如下内容。

说明

- 如果要配置权限的用户已经在sudoers文件中存在，则只需在该用户下增加权限。
- 如果要配置权限的用户在sudoers文件中不存在，则增加用户后，为其配置权限。

username ALL=(ALL:ALL) NOPASSWD:/usr/bin/perf trace -T ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux**/toolkit/tools/system-profiling/malluma/plugins/collector/exe/time_gap, /usr/bin/perf trace -T --syscalls -p [0-9]*, /usr/bin/ltrace -ttt -T -f -e pthread_* -p [0-9]*, /usr/sbin/iotop -t -d 0.02 -b -P -p [0-9]*, /bin/bash ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux**/toolkit/tools/system-profiling/script/../malluma/install/restart.sh, /bin/bash ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux**/toolkit/tools/system-profiling/script/../malluma/install/stop.sh, /bin/bash ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux**/toolkit/tools/system-profiling/malluma/plugins/collector/exe/kill_process.sh

其中{arch}请替换为实际架构（x86_64或arm64）。

步骤4 添加完成后，执行**:wq!**保存文件。

步骤5 执行以下命令取消“/etc/sudoers”文件的写权限：

```
chmod u-w /etc/sudoers
```

----结束

11.3 使用指南

11.3.1 启动 System Profiling

须知

启动System Profiling前，需要保证没有高CPU占用率的进程在运行，否则可能造成System Profiling数据采集执行不准确（例如Host侧CPU数据采集不完整）。

步骤1 用MindStudio安装用户进入System Profiling脚本目录。

```
cd Ascend-cann-toolkit安装目录/ascend-toolkit/{version}/{arch}-linux/toolkit/tools/system-profiling/script
```

步骤2 执行**./start.sh**或者**./start.sh debug**命令启动后端服务。

说明

- 启动后台服务的用户不能配置代理，如果配置代理需要手动关闭。
- 只能由一个用户启动后端服务并进行数据采集，暂不支持多个用户同时采集。
- 如果用户修改了*Ascend-cann-toolkit安装目录*/**ascend-toolkit/***{version}*/*{arch}*-**linux/toolkit/tools/system-profiling/config**目录下的**env.conf**文件，需要执行如下命令重启后端服务，配置文件才能生效。

  ./stop.sh结束后端服务进程。

  ./start.sh或者**./start.sh debug**启动后端服务。

步骤3 进入System Profiling。

- 对新项目进行System Profiling：选择“Ascend > System Profiling > New Project”。
- 选择“Ascend > System Profiling > Profile”。
- MindStudio欢迎界面单击“Welcome System Profiling”，如**图11-1**所示。

图 11-1 欢迎界面入口

说明

后两种方式会直接进入System Profiling调优分析界面，可以浏览历史报告，无需进行后续参数配置步骤。

步骤4 配置profiling工程属性。

- Project Name：工程名称
- Project Location：工程文件路径

步骤5 单击“Next”，进入profiling可执行文件属性配置界面。

表 11-1 参数说明

参数	参数说明
Run Mode	● Remote Run：远程执行System Profiling。 ● Local Run：本地执行System Profiling。
SSH Connection	SSH连接配置，Run Mode选择Remote Run时需要配置。 单击下拉框可以选择已有SSH连接。 单击新增SSH连接，具体配置方法参见**13.1 SSH连接管理**。其中SSH tunnel参数的远程端口请设置为 ***Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*/*{arch}*-linux** **/toolkit/tools/system-profiling/config**目录下的env.conf文件中的server_port参数值；设置的本地端口请确保未被占用。 **说明** 选择SSH连接后可根据提示判断是否成功连接。

版权所有 © 华为技术有限公司

参数	参数说明
App Project Path	执行System Profiling目标工程目录，Run Mode选择Remote Run时需要配置。
Executable File	执行System Profiling目标工程的可执行文件目录。
Command Arguments	用户APP的执行参数，由用户自行配置，参数之间用空格分隔，默认为空。
Device ID	与用户APP绑定，用户根据APP实际运行的Device指定其ID，DeviceID取值范围为0~63。如输入多个设备中间用半角逗号隔开。
Environment Variables	配置环境变量参数，单击右侧配置图标进行环境变量配置。

步骤6 单击“Next”，进入profiling参数配置界面。

表 11-2 参数配置说明

类别	参数	配置说明
AI Core Profiling	Task-based	● 以算子为单位对AI Core的执行行为进行profiling。 ● 有算子的timeline数据展示。 ● summary页面无AI Core利用率折线图展示。 ● 默认勾选。
	Sample-based	● 基于采样方式对AI Core的执行行为进行profiling。 ● 无法获取算子的Timeline。 ● summary页面有AI Core利用率折线图展示。
	Frequency	采样频率，单位为Hz，默认频率为10Hz。
API Trace	ACL API	Ascend Computing Language API调用，默认勾选不能取消。
	Runtime API	ACL底层API调用，默认勾选不能取消。
	OS Runtime API	系统运行时函数库API和Pthreads API的调用，默认勾选。 **说明** 采集OS Runtime API调用数据需要安装第三方开源工具perf和ltrace，使用开源工具ltrace采集OS Runtime API调用数据会导致CPU占用率过高，其与应用工程的pthread加解锁相关，会影响进程运行速度。

类别	参数	配置说明
Device System Sampling	DDR Frequency	采集DDR数据。默认勾选，可以更改采样频率，默认频率为50Hz。
Host CPU Sampling	CPU	采集Host侧CPU资源利用率信息。默认勾选。
	Memory	采集Host侧Memory资源利用率信息。
	Disk	采集Host侧Disk资源利用率信息。需要安装好iotop工具。
	Network	采集Host侧Network资源利用率信息。

步骤7 单击“Start”，启动System Profiling。

说明

启动后，在弹出的进度条窗口中，可以进行Cancel操作。单击Cancel可以取消System Profiling采集过程。

----结束

11.3.2 Project Explorer

展示创建的工程和生成的profiling报告。

创建新工程

步骤1 右键单击“空白处 > New Project”。

步骤2 参考**步骤4**到**步骤7**完成创建。

----结束

已有工程中创建新的报告

步骤1 右键单击“工程名称 > Profiling Configuration”。

步骤2 参考**步骤5**配置可执行文件属性。

步骤3 参考**步骤6**配置profiling参数。

步骤4 单击“Start”，生成新的profiling报告。

----结束

打开未展示的报告

步骤1 右键单击“空白处 > Open...”。

步骤2 选择目标报告，单击Open。

----结束

删除工程

步骤1 右键单击“工程名称 > Delete Project”。

步骤2 在弹出的对话框中单击“Yes”。

----结束

说明

此操作会删除目标工程和工程目录下所有报告。

打开报告

步骤1 打开工程目录。

步骤2 左键双击需要打开的报告或者右键单击“报告名称 > Open Report”。

----结束

另存报告

步骤1 打开工程目录。

步骤2 右键单击“报告名称 > Save As”。

步骤3 选择目标路径，单击“Save”保存报告。

----结束

删除报告

步骤1 打开工程目录。

步骤2 右键单击“报告名称 > Delete Report”。

步骤3 在弹出的对话框中单击“Yes”。

----结束

刷新导航栏

右键单击“空白处 > Refresh”。刷新后可以将导航栏状态刷新到最新状态，所有已经展开的工程目录会收起。

11.3.3 TimeLine View

TimeLine View包含左侧导航窗格和右侧图形化窗格。导航窗格显示各个Timeline的名称，以及各个Timeline之间的从属关系。图形化窗格对应导航窗格，逐行对Timeline进行图形化展现。

说明

- 将鼠标移动到具体采样点，可以查看对应采样点具体分析数据。
- 在左侧导航窗格中右键标签名，选择“Show in Event View”，可以在Event View界面中看到对应选项Timeline的顺序执行信息。
- 在Timeline中可以查看对应的API或操作的名称。
- 同一线程中有多个Os Runtime API同时执行时会分行显示。
- 同一Ascend Compute的同一stream中有多个AI Core task同时执行时会分行显示。
- 当选中Timeline中某个时间点时，按住Ctrl键并向上/下滚动鼠标滑轮或点击视图右上角的 / ，实现Timeline视图的放大/缩小。点击视图右上角的 实现视图复位。

Timeline 颜色配置

在MindStudio菜单栏选择“Ascend > System Profiling > Timeline Color”。Timeline颜色配置窗口如图11-2所示。

图 11-2 Timeline 颜色配置

可以根据API执行时间自定义配置Timeline中的颜色显示。

- 勾选Accumulative：在不同时间被调用的相同API执行时间做累加计算。
- 不勾选Accumulative：在不同时间被调用的相同API执行时间分别计算。

例如图11-2中所示，为默认配置，执行时间大于等于5%且小于10%的API在Timeline中显示为黄色，执行时间大于等于10%的API显示为红色。

说明

需要关闭并重新打开report才能应用新的颜色配置。

CPU Timeline

Host侧CPU资源利用率数据展示。

表 11-3 CPU Timeline 参数解释

参数	参数解释
Start Time	对应CPU开始运行时间。
End Time	对应CPU结束运行时间。
Duration	对应CPU运行持续时间。
Usage	对应CPU利用率。

Memory Timeline

Host侧Memory资源利用率数据展示。

表 11-4 Memory Timeline 参数解释

参数	参数解释
Start Time	对应Memory开始运行时间。
End Time	对应Memory结束运行时间。
Duration	对应Memory运行持续时间。
Usage	对应Memory利用率。

Disk Timeline

Host侧Disk资源利用率数据展示。

表 11-5 Disk Timeline 参数解释

参数	参数解释
Start Time	对应Disk开始运行时间。
End Time	对应Disk结束运行时间。
Duration	对应Disk运行持续时间。
Usage	对应Disk利用率。

Network Timeline

Host侧Network资源利用率数据展示。

表 11-6 Network Timeline 参数解释

参数	参数解释
Start Time	对应Network开始运行时间。
End Time	对应Network结束运行时间。
Duration	对应Network运行持续时间。
Usage	对应Network利用率。

Process Timeline

各个进程中所有线程下Os Runtime、ACL API、Runtime API的运行时间数据展示。

表 11-7 参数解释

参数	参数解释
Start Time	对应线程开始运行时间。
End Time	对应线程结束运行时间。
Duration	对应线程运行持续时间。
Name	对应线程中的API名称。

NPU Timeline

算子运行数据展示。

说明

Profiling参数配置中的AI Core Profiling选择Sample-based采集方式没有算子的Timeline（对应“Ascend Compute”）数据。

表 11-8 Ascend Compute 参数解释

参数	参数解释
Start Time	对应算子开始运行时间。
End Time	对应算子结束运行时间。
Duration	对应算子运行持续时间。
Status	对应算子运行状态。
Task_type	对应算子任务类型。
Kernel_function	对应算子函数。
Streamid	对应算子的stream ID。

参数	参数解释
Opname	对应算子名称。
Task	对应算子task ID。

11.3.4 Event View

在TimeLine View界面的左侧导航窗格中右键标签名，选择“Show in Event View”，可以在Event View中看到对应选项Timeline的顺序执行信息，操作如图所示。

图 11-3 Show in Event View

可查看项目包括：

- Process > Thread > OS Runtime
- Process > Thread > ACL API > model
- Process > Thread > ACL API > runtime
- Process > Thread > Runtime API > runtime
- NPU > Ascend Compute > stream > AI Core task

Os Runtime

表 11-9 参数解释

参数	参数解释
ID	Os Runtime的ID。
Name	Os Runtime的名称。

版权所有 © 华为技术有限公司

参数	参数解释
Start(s)	Os Runtime开始运行时间。
End(s)	Os Runtime结束运行时间。
Process ID	Os Runtime对应进程ID。
Thread ID	Os Runtime对应线程ID。

ACL API model

表 11-10 参数解释

参数	参数解释
ID	ACL API model的ID。
Name	ACL API model的名称。
Start(s)	ACL API model开始运行时间。
End(s)	ACL API model结束运行时间。
Process ID	ACL API model对应进程ID。
Thread ID	ACL API model对应线程ID。

ACL API runtime

表 11-11 参数解释

参数	参数解释
ID	ACL API runtime的ID。
Name	ACL API runtime的名称。
Start(s)	ACL API runtime开始运行时间。
End(s)	ACL API runtime结束运行时间。
Process ID	ACL API runtime对应进程ID。
Thread ID	ACL API runtime对应线程ID。

Runtime API runtime

表 11-12 参数解释

参数	参数解释
ID	Runtime API的ID。
Name	Runtime API的名称。
Start(s)	Runtime API开始运行时间。
End(s)	Runtime API结束运行时间。
Process ID	Runtime API对应进程ID。
Thread ID	Runtime API对应线程ID。

AI Core task

表 11-13 参数解释

参数	参数解释
ID	AI Core task的ID。
Device ID	AI Core task运行所在设备ID。
Op Name	AI Core task的算子名称。
Start(us)	AI Core task的开始运行时间。
End(us)	AI Core task的结束运行时间。
Status	AI Core task的运行状态。
Task Type	AI Core task的任务类型。
Kernel Function	AI Core task的kernel函数名称。
Stream ID	AI Core task的stream ID。
Task ID	AI Core task的task ID。

11.3.5 Statistics

单击下方窗口的Statistics，通过左上角下拉框选择不同类别的数据，可以查看ACL API、OS Runtime API、Runtime API和OP Statistics调用情况数据。

ACL API、OS Runtime API、Runtime API

表 11-14 参数说明

参数	参数说明
Process ID	对应调用API所在进程ID。
Thread ID	对应调用API所在线程ID。
API Name	对应调用API名称。
Time(%)	对应调用API总时间占比。
Time(us)	对应调用API时间长度，单位为us。 可以点击字段旁边的三角号根据此项进行降序或升序排列。
Count	对应API调用次数。
Avg(us)	对应API单次调用平均时间，单位为us。
Max(us)	对应API单次调用最长时间，单位为us。
Min(us)	对应API单次调用最短时间，单位为us。

OPs

可以点击Device下拉框，选择查看不同Device的算子调用数据。

表 11-15 OP Statistics 参数说明

参数	参数说明
Task ID	Task ID，用于识别task。
Stream ID	Stream ID，用于识别stream。
Op Name	算子名称。
Time(%)	百分比耗时。
Time(us)	时间单位耗时，单位us。
Count	算子调用次数。
Avg(us)	耗时平均值，单位us。
Max(us)	耗时最大值，单位us。
Min(us)	耗时最小值，单位us。

11.3.6 AI Core Metrics

单击Device ID旁边的下拉框，可以查看不同device ID的AI Core Metrics数据。分为Task-based和Sample-based两种采集方式。

采用Task-based采集方式请参考表11-16，采用Sample-based采集方式请参考表11-17。

表 11-16 Task-based

参数	参数说明
Task ID	Task ID，用于识别task。
Stream ID	Stream ID，用于识别stream。
Op Name	算子名称，用于识别算子。
AI Core Time(us)	AI Core运行时间，单位us。
Vec Ratio	vector类型指令（向量类运算指令）的cycle数在所有指令的cycle数中的占用比。
Mac Ratio	cube类型指令（矩阵类运算指令）的cycle数在所有指令的cycle数中的占用比。
Scalar Ratio	scalar类型指令（标量类运算指令）的cycle数在所有指令的cycle数中的占用比。
Mte1 Ratio	mte1类型指令（L1->L0A/L0B搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Mte2 Ratio	mte2类型指令（DDR->AI Core搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Mte3 Ratio	mte3类型指令（AI Core->DDR搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Icache Miss Rate	icache缺失率，即未命中icache，数值越小越好。
Memory Bound	用于识别AI Core执行算子计算过程是否存在内存瓶颈，由mte2_ratio/max(mac_ratio, vec_ratio)计算得出。计算结果小于1表示没有内存瓶颈；计算结果大于1表示有内存瓶颈，且数值越大瓶颈越严重。

表 11-17 Sample-based

参数	参数说明
Core ID	AI Core ID，用于识别AI Core。
Vec Ratio	vector类型指令（向量类运算指令）的cycle数在所有指令的cycle数中的占用比。

参数	参数说明
Mac Ratio	cube类型指令（矩阵类运算指令）的cycle数在所有指令的cycle数中的占用比。
Scalar Ratio	scalar类型指令（标量类运算指令）的cycle数在所有指令的cycle数中的占用比。
Mte1 Ratio	mte1类型指令（L1->L0A/L0B搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Mte2 Ratio	mte2类型指令（DDR->AI Core搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Mte3 Ratio	mte3类型指令（AI Core->DDR搬运类指令）的cycle数在所有指令的cycle数中的占用比。
Icache Miss Rate	icache缺失率，即未命中icache，数值越小越好。
Memory Bound	用于识别AI Core执行算子计算过程是否存在内存瓶颈，由mte2_ratio/max(mac_ratio, vec_ratio)计算得出。计算结果小于1表示没有内存瓶颈；计算结果大于1表示有内存瓶颈，且数值越大瓶颈越严重。

11.3.7 Analysis Summary

在“Analysis Summary”模块中，选择需要查看的Device ID后，可以查看对应device的详细性能分析信息。

图 11-4 Analysis Summary 视图

Analysis Summary | Device ID 3

Profiling Info

Result Size	Collection Start Time	Collection End Time
0.741 MB	1608794908.912922	1608794910.227956

Host System Info

Host Operating System	Host Computer Name
Linux-4.15.0-29-generic-#31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018	ubuntu

Host CPU Info

CPU ID	Name	Type	Logical CPU Count
CPU0	GenuineIntel	Intel(R) Xeon(R) CPU E7-8880 v3 @ 2.30GHz	36
CPU1	GenuineIntel	Intel(R) Xeon(R) CPU E7-8880 v3 @ 2.30GHz	36
CPU2	GenuineIntel	Intel(R) Xeon(R) CPU E7-8880 v3 @ 2.30GHz	36
CPU3	GenuineIntel	Intel(R) Xeon(R) CPU E7-8880 v3 @ 2.30GHz	36

Device Info

Control CPU Type	Control CPU Number	TS CPU Number	AI CPU Number	AI Core Number
ARMv8_Cortex_A55	4	1	4	2

DDR

Metric	Read	Write
Average	218.892915	167.64288

OP Statistics

Task ID	Stream ID	Op Name	Time(%)	Time(us)	Count	Avg(us)	Max(us)	Min(us)
1	514	N/A	18.2296999999999...	3482.406	1	3482.406	3482.406	3482.406
2	514	N/A	17.8428	3408.501	1	3408.501	3408.501	3408.501
53	517	res5b_branch2bres...	2.95300000000000...	564.105	2	282.052	282.808	281.297
56	517	res5c_branch2bres...	2.87719999999999...	549.627	2	274.813	276.09	273.537
54	517	res5b_branch2cres...	2.67730000000000...	511.45	2	255.725	256.923	254.527
50	517	res5a_branch2c	2.6604	508.221	2	254.11	257.496	250.725
49	517	res5a_branch2bres...	2.6356	503.481	2	251.74	259.37	244.111

Collection Info

表 11-18 收集信息

参数	参数解释
Result Size	结果文件大小。
Collection Start Time	收集开始时间。
Collection End Time	收集结束时间。

Host System Info

表 11-19 Host 系统信息

参数	参数解释
Host Operating System	Host侧操作系统信息。
Host Computer Name	Host侧电脑名称。

Host CPU Info

表 11-20 Host 侧 CPU 信息

参数	参数解释
CPU ID	CPU ID。
Name	CPU名称。
Type	CPU型号。
Logical CPU Count	逻辑CPU数量。

Device Info

表 11-21 Device 信息

参数	参数解释
Control CPU Type	Control CPU型号。
Control CPU Number	Control CPU数量。
TS CPU Number	TS CPU数量。
AI CPU Number	AI CPU数量。

参数	参数解释
AI Core Number	AI Core 数量。

DDR 数据

表 11-22 DDR 参数说明

Metric	Read	Write
BandWidth(MB/s)	读带宽，单位为MB/s。	写带宽，单位为MB/s。

OP Statistics 数据

表 11-23 OP Statistics 参数说明

参数	参数说明
Task ID	Task ID，用于识别task。
Stream ID	Stream ID，用于识别stream。
Op Name	算子名称。
Time(%)	百分比耗时。
Time(us)	时间单位耗时，单位us。
Count	算子调用次数。
Avg(us)	耗时平均值，单位us。
Max(us)	耗时最大值，单位us。
Min(us)	耗时最小值，单位us。

AI Core 利用率

AI Core利用率通过折线图方式呈现（AI Core Profiling参数选择Sample-based才会展示）。

图 11-5 AI Core 利用率

版权所有 © 华为技术有限公司

12 AI Core Error 分析工具

12.1 功能介绍
12.2 总体流程
12.3 数据准备
12.4 通过工具生成解析数据
12.5 问题分析和定位

12.1 功能介绍

在执行训练和推理发现AI Core Error问题时，使用AI Core Error Analyzer工具可以自动快速准确地收集定位AI Core Error问题所需的关键信息，提升开发者对AI Core Error的排查效率。

说明

- 该工具当前不支持在容器中部署使用。
- 该工具当前仅支持本地分析使用，即部署该工具的环境应该和日志所在环境为同一环境。
- 在推理场景下，默认使用root用户运行AI Core Error分析工具，如需使用普通用户运行，请参考《**日志参考(推理)**》手册中的“附录 > msnpureport工具使用”章节，使用root用户修改目录属性，并且给普通用户添加执行msnpureport工具的权限。

版权所有 © 华为技术有限公司

12.2 总体流程

推理场景

图 12-1 推理流程

步骤1 执行推理时，发现了AI Core Error问题。

步骤2 配置支持AI Core Error分析工具相关参数，包括支持异常算子代码行号映射的op_debug_level参数以及debug_dir参数。

步骤3 重新执行模型转换，生成指令映射文件。

步骤4 使用AI Core Error分析工具解析问题。

----结束

版权所有 © 华为技术有限公司

训练场景

图 12-2 训练流程

步骤1 进行模型训练时，发现了AI Core Error问题。

步骤2 配置支持AI Core Error分析工具相关参数，包括如下参数：

- 支持异常算子代码行号映射的op_debug_level参数。
- 支持dump异常算子信息的enable_exception_dump参数。

步骤3 重新执行模型训练脚本，生成指令映射文件和异常算子的dump文件。

步骤4 使用AI Core Error分析工具解析问题。

----结束

12.3 数据准备

训练场景

1. 当训练过程中发现AI Core Error问题时，用户可参考如下步骤配置op_debug_level和enable_exception_dump参数。
   - Estimator模式下，开启op_debug_level和enable_exception_dump：

```
from npu_bridge.estimator.npu.npu_config import NPURunConfig
from npu_bridge.estimator.npu.npu_config import DumpConfig

session_config=tf.ConfigProto()

config = NPURunConfig(
  op_debug_level = 2,          //开启算子debug功能。
  session_config=session_config,
  enable_exception_dump=1      //Dump AI Core Error算子的输入和输出信息，dump信息生成
```

```
在当前脚本执行目录。不支持dump动态shape算子。
  )
```

– sess.run模式下，开启op_debug_level和enable_exception_dump：

```
import tensorflow as tf
from npu_bridge.estimator import npu_ops
from tensorflow.core.protobuf.rewriter_config_pb2 import RewriterConfig

config = tf.ConfigProto()

custom_op =  config.graph_options.rewrite_options.custom_optimizers.add()
custom_op.name =  "NpuOptimizer"
custom_op.parameter_map["use_off_line"].b = True
custom_op.parameter_map["enable_exception_dump"].i = 1  //Dump AI Core Error算子的输入和输出信息，dump信息生成在当前脚本执行目录。不支持dump动态shape算子。
custom_op.parameter_map["op_debug_level"].i = 2        //开启算子debug功能。
config.graph_options.rewrite_options.remapping = RewriterConfig.OFF  #关闭remap开关

with tf.Session(config=config) as sess:
  print(sess.run(cost))
```

2. 重新执行训练后，会在训练执行目录下生成指令映射文件和异常算子dump文件。

推理场景

需要在模型转换时，完成op_debug_level和debug_dir参数配置。流程如下：

步骤1 启动Model Converter。

步骤2 单击Advanced Options下拉框。

步骤3 将Additional Arguments置于开启状态，填入“op_debug_level”参数（例如：--op_debug_level=2）和“debug_dir”参数（生成debug文件的路径，该路径为工程下的指定路径）。

图 12-3 配置参数

----结束

op_debug_level 参数说明

表 12-1 op_debug_level 参数取值说明

取值	说明
0	不开启算子debug功能，默认为0。
1	开启算子debug功能，在训练脚本执行目录下的kernel_meta文件夹中生成TBE指令映射文件（算子cce文件*.cce、python-cce映射文件*_loc.json、.o和.json文件），用于后续工具进行AI Core Error问题定位。
2	开启算子debug功能，在训练脚本执行目录下的kernel_meta文件夹中生成TBE指令映射文件（算子cce文件*.cce、python-cce映射文件*_loc.json、.o和.json文件），并关闭编译优化开关（开启ccec编译器选项-O0-g），用于后续工具进行AI Core Error问题定位，可定位到出错算子的代码行号。

版权所有 © 华为技术有限公司

12.4 通过工具生成解析数据

启动工具

步骤1 单击工具栏菜单“Ascend > AI Core Error Analyzer”，启动AI Core Error Analyzer。

步骤2 AI Core Error Analyzer参数配置。

Compile Path：推理场景为模型转换生成的debug文件路径，请参见“数据准备”章节中的**步骤3**。训练场景为脚本执行路径。用于存储算子编译中间生成的输出件和pbtxt，一般为“kernel_meta”文件的父路径（例如：~/model_convert）。

图 12-4 参数配置

步骤3 单击“Analyze”，在“Output”窗口查看分析结果。

----结束

生成结果文件

所有AI Core Error Analyzer生成的文件会保存在当前IDE路径下，当前IDE路径格式一般为“info_*timeline*”（例如“info_20200903114406”）。生成的结果文件请以实际情况为准，以下示例仅供参考。

- 推理场景（示例）：
  - Ascend EP场景：

```
├──── aicore_error
│    ├────aicerr_out
│        ├────info_xxxxx
│            ├────aicerror_xxxxx  //AI Core Error分析工具生成文件
│                ├────info.txt    //AI Core Error工具解析结果汇总文件
│                ├────te_transdata_xxxx.o
│                ├────te_transdata_xxxx.o.txt   //反编译文件
│            ├────collection    //异常算子涉及文件
│                ├────compil
│                    ├────kernel_meta
│                        ├────cce文件
│                        ├────json文件
│                        ├────loc.json文件
│                        ├────.o文件
│                    ├────ge_proto_xxxx_Build.txt
│                ├────dump    //搬运的dump文件
│                ├────log    //Host侧日志
│                ├────xxxxx   //时间信息
│            ├────error.log    //收集了日志目录下所有ERROR级别日志信息
│            ├────README.txt
│    ├────npu_report
│        ├────xxxxx   //时间信息
│            ├────hisi_logs    //黑匣子错误信息
│            ├────message    //Device侧操作系统日志
│            ├────slog
│            ├────stackcore
```


版权所有 © 华为技术有限公司

– Ascend RC场景：

```
├──── aicore_error
│    ├────aicerr_out
│        ├────info_xxxxx
│            ├────aicerror_xxxxx  //AI Core Error分析工具生成文件
│                ├────info.txt    //AI Core Error工具解析结果汇总文件
│                ├────te_transdata_xxxx.o
│                ├────te_transdata_xxxx.o.txt   //反编译文件
│            ├────collection    //异常算子涉及文件
│                ├────compil
│                    ├────kernel_meta
│                        ├────cce文件
│                        ├────json文件
│                        ├────loc.json文件
│                        ├────.o文件
│                    ├────ge_proto_xxxx_Build.txt
│                ├────dump    //搬运的dump文件
│                ├────xxxxx
│            ├────error.log    //收集了日志目录下所有ERROR级别日志信息
│            ├────README.txt
```

- 训练场景（示例）：

```
├──── aicore_error
│    ├────aicerr_out
│        ├────info_xxxxx
│            ├────collection    //异常算子涉及文件
│                ├────log    //Host侧日志
│                    ├────xxxx    //进程ID
│                ├────xxxx    //时间信息
│                    ├────hisi_logs    //黑匣子错误信息
│                    ├────slog
│    ├────npu_report
│        ├────xxxxx    //时间信息
│            ├────hisi_logs    //黑匣子错误信息
│            ├────message    //Device侧操作系统日志
│            ├────slog
│            ├────stackcore
```

12.5 问题分析和定位

用户可以直接通过run窗口的日志进行问题分析和定位。关键信息说明：

```
*******************Root cause conclusion********************
#符合已知故障模式的问题会列出对问题根因的判断。

**********************1. Basic information*******************
#AI Core error发生时设备的基本信息。
#kernel name：算子名称。
#op address: 算子代码在ddr内存的地址。
#args address：算子参数在ddr内存的地址。
error time   : 2020-08-26-11:24:07
device id    : 0
core id      : 0
task id      : 60
stream id    : 517
node name    : trans_TransData_167
kernel name  : te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb
op address   : 0x101000120000
args address : 0X101000053000

**********************2. AICERROR code**********************
#AI Core error错误码及解释。
code  : 0x10

CCU_ERR_INFO: 0xb166486200070074
    ccu_err_addr bit[22:8]=000011100000000  meaning:CCU Error Address [17:3]  approximate:0x3800

**********************3. Instructions***********************
```

```
#异常指令。
start   pc   : 0x101000120000
current pc   : 0x1010001201e0

Error occured most likely at line: 1d0

/{当前IDE路径}/aicerror_xxxx/te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.o.txt:1d0
/{当前IDE路径}/collection/compile/kernel_meta/te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.cce:
32    //出错算子的cce代码行号
/{python脚本文件路径}/nz_2_nd.py:4486     //出错算子的python代码行号

related instructions (error occured before the mark *):

     1bc: <not available>
     1c0: <not available>
     1c4: <not available>
     1c8: <not available>
     1cc: <not available>
     1d0: <not available>
     1d4: <not available>
     1d8: <not available>
     1dc: <not available>
*    1e0: <not available>

For complete instructions, please view /{当前IDE路径}/aicerror_xxxx/
te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.o.txt

****************4. Input and output of node*******************
#输入输出信息。
#从GE的IMAS日志中解析出输入输出地址，从build图中解析出size。
#如果有内存零拷贝情况，日志中可以解析出新地址new addr。
#如果地址不在RTS申请日志的记录范围内，会给出 OVERFLOW 标记。
#如果收集了device内存数据，会检测数据中是否有 NaN/INF ，但只有在device挂起后收集到的数据才是准确
的。
#如果检测到kernel函数中输入输出个数与此处个数不相等，会给出 WARNING，很有可能发生了GE给的参数与
算子处理的参数错位问题。
input[0]   addr: 0x100801126600   size: 32288
output[0]   addr: 0x100801157c00   size: 2048

***********************5. Op in graph*************************
#出错算子信息。
#从build图中截取op信息，方便查看。

***********************6. Dump info***************************
#异常算子dump文件信息。
#训练场景独有。
```

13 操作指南

13.1 SSH连接管理
13.2 工程管理
13.3 运行管理
13.4 设备管理
13.5 ADK管理
13.6 日志管理
13.7 常用快捷键
13.8 常用原生功能

13.1 SSH 连接管理

说明

为了安全起见，建议对SSH连接到的服务器进行以下加固操作：

- Linux系统用户的口令加密算法，推荐使用SHA256或者SHA512。
- SSH服务端配置文件中，对以下配置项进行加固：
  - MACs：指定允许在SSH-2中使用哪些消息摘要算法来进行数据校验，多个算法之间使用逗号分隔。算法建议使用hmac-sha2-512，hmac-sha2-256。
  - Ciphers：指定SSH-2允许使用的加密算法，多个算法之间使用逗号分隔。算法建议使用aes128-ctr，aes192-ctr，aes256-ctr。
  - PermitRootLogin：是否允许root登录。建议禁止root用户可以直接登录。

13.1.1 概述

配置入口

有如下三种方式进入SSH连接配置：

- 在菜单栏选择Ascend > SSH Configurations。
- 在菜单栏选择File > Settings... > Tools > SSH Configurations。

- 欢迎界面选择Configure > Settings。

图 13-1 SSH 配置

说明

- 通过前两种方式进入（打开工程状态），SSH连接配置界面上方会显示For current project和Visible only for this project复选框。
  - 勾选Visible only for this project：配置的SSH连接只对本项目可见。
  - 不选Visible only for this project：配置的SSH连接对所有项目可见。
- SSH连接配置界面左侧展示SSH连接配置项别名，如果没有别名，显示格式为<username>@<host>:<port>，单击要查看的SSH连接配置项，可以在右侧详细面板中查看参数配置。

参数及图标说明

表 13-1 参数及图标说明

参数及图标	说明
Host	连接目标IP地址。
Port	连接目标IP地址的端口号。
User name	登录目标IP的用户名。

参数及图标	说明
Authentication type	身份验证方式。有两种方式可选： ● PASSWORD：通过密码验证身份。 ● KEY_PAIR：通过密钥验证身份，可以自行参考通用方式申请SSH密钥。 **说明** 对于EulerOS2.8 aarch架构，使用KEY_PAIR方式来通过密钥验证身份时，需要使用**ssh-keygen -m PEM**或者**ssh-keygen -t rsa -m PEM**命令来申请SSH密钥。
Password	Authentication type选择验证方式为PASSWORD时显示，需要填写对应用户名的密码。勾选Save password复选框可以保存密码。
Private key file	Authentication type选择验证方式为KEY_PAIR时显示，选择本地私钥文件。
Passphrase	Authentication type选择验证方式为KEY_PAIR时显示，需要填写对应密语。勾选Save passphrase复选框可以保存密语。
Test Connection	单击该按钮测试连接是否成功。
SSH tunnel	本地端口与远程端口的映射，格式为<local port>:<remote port>。
Remote work path	远程环境工作目录。
+	新增SSH连接配置。
−	删除SSH连接配置，选中需要删除的SSH连接配置，单击该按钮删除。
	另存SSH连接配置，选中需要另存的SSH连接配置，单击该按钮另存。
	修改SSH连接别名，选中需要修改的SSH连接配置，单击该按钮进行别名修改。
OK	保存SSH配置，单击OK按钮保存、应用SSH连接当前配置，并关闭配置界面。
Cancel	取消当前配置，单击Cancel按钮不保存当前配置，并关闭配置界面。
Apply	应用SSH配置，单击Apply按钮保存并应用SSH连接当前配置。

13.1.2 新增 SSH 连接

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 单击界面左侧导航栏上方+按钮。

步骤3 根据参数及图标说明配置参数后，单击“OK”或“Apply”。

----结束

13.1.3 删除 SSH 连接

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 选择要删除的SSH连接配置项，单击界面左侧导航栏上方按钮。

步骤3 单击“OK”或“Apply”。

----结束

13.1.4 修改 SSH 连接

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 选择左侧导航栏中需要修改的SSH连接配置，可单击界面左侧导航栏上方按钮修改名称。

步骤3 根据参数及图标说明修改参数后，单击“OK”或“Apply”。

----结束

13.1.5 复制 SSH 连接

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 选择左侧导航栏中需要复制的SSH连接配置，单击界面左侧导航栏上方按钮。

步骤3 输入新的SSH名称，可根据参数及图标说明修改参数后，单击“OK”或“Apply”。

----结束

13.1.6 加密 SSH 密码

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 填写对应用户名的密码，勾选“Save password”复选框，单击“Apply”，弹出密码加密窗口。

图 13-2 密码加密窗口

- Password：输入加密密码。
- Confirm：再次输入加密密码。

步骤3 选择是否进行密码加密。

- 取消密码保存：直接单击图13-2中的“Cancel”，取消对应用户名的密码保存。
- 进行密码加密：
  a. 填写图13-2中的“Password”和“Confirm”参数。
  b. 单击“Continue”，完成密码加密。
- 不进行密码加密：
  a. 不填写“Password”和“Confirm”参数，直接单击图13-2中的“Continue”，弹出以下提示框。

  图 13-3 不进行密码加密提示框

  b. 单击“Continue”，不进行密码加密。

步骤4 单击“OK”。

----结束

13.1.7 修改 SSH 密码保存方式

步骤1 参见配置入口进入SSH连接配置界面。

步骤2 进入密码保存方式界面有如下方式：

- 在左边的导航栏选择“Appearance & Behavior > System Settings > Passwords”。
- 在搜索栏中输入“Passwords”。

图 13-4 密码保存方式界面

步骤3 选择密码保存方式。

- In native Keychain
- In KeePass
- Do not save，forget passwords after restart

说明

- 三种密码保存方式的详细说明请参见《**IntelliJ IDEA使用说明**》中的“**Passwords**”章节。
- 由于GNOME Keyring工具安装不成功等原因导致“In native Keychain”密码保存方式无法使用时，建议使用“In KeePass”密码保存方式。

步骤4 单击“OK”或“Apply”。

----结束

13.2 工程管理

13.2.1 打开/关闭/移除

打开工程

如果用户之前创建过工程，则可以通过如下两种界面方式打开之前创建的工程。

- 在工程打开界面，依次选择"File > Open...",选择工程所在路径后，弹出**图13-5**所示界面。

图 13-5 打开已有工程

选中相应工程，单击“OK”。

- 在工程欢迎界面，单击“Open or Import”，在弹出界面中选择工程所在路径后，进入图13-5所示界面，选择相应工程，单击OK。

若弹出如图13-6所示界面，说明已存在工程中的ADK版本与当前MindStudio中的ADK版本不匹配，请单击OK，然后在弹出的界面中重新选择当前**ADK Version**即可，然后单击**OK**关闭配置窗口。

图 13-6 工程打开异常提示

打开最近工程

在已经有Project关闭的情况下，可以通过如下两种方式，打开最近使用的工程，方法如下：

1. 依次单击菜单栏的“File > Open Recent”，在右侧弹出页面中选择想要打开的工程，如图13-7所示中的工程。

图 13-7 Open Project

单击相应工程后弹出如图13-8所示页面。

图 13-8 工程打开提示

- 选择“This Window”，则直接在当前工作窗口打开选中工程。
- 选择“New Window”，则新建一个工作窗口打开选中的工程。

2. 依次单击菜单栏的 “File > Open Recent > Manage Projects…”，在弹出页面中选择想要打开的工程，选中后会在新窗口中打开相应工程。

关闭工程

单击File菜单中的“Close Project”，会把当前已经打开的工程关闭。如果关闭的是最后一个工程，系统会弹出如图13-9所示界面。您可以根据需要选择重新打开已关闭工程、切换到其他工程或者单击右侧区域“Create new project”新建工程。

图 13-9 欢迎页面

移除工程

工程关闭后，如果您不想继续使用相关工程，可以选择将其移除。移除工程有两种方法：

- 在图13-9所示界面中，选中想移除的工程，单击右侧的 或者右击选择“Remove Selected from Welcome Screen”。
- 单击File菜单中的“Open Recent > Manage Projects...”，在弹出页面中选中想移除的工程，单击右侧的 移除。

图 13-10 移除工程

13.2.2 新建/删除文件

新建文件

在Projects视图中，选中相应工程，单击鼠标右键选择“New”（或者在File菜单中选择New），在弹出的子菜单中选择需要创建的文件类型或直接单击File，在打开的对话框中输入文件名（不需要输入文件类型对应的后缀名）。示例：直接单击File后弹出的页面如图13-11所示。

图 13-11 输入新建文件名

若已经选择文件类型，则输入文件名后单击“OK”即可；若为图13-11所示新建文件，输入文件名后回车，会弹出选择文件类型页面，如图13-12所示，用户可以根据需要选择新建文件的类型。

图 13-12 选择新建文件类型

选择相关类型后，单击“OK”。

删除文件

选中文件，鼠标右键，在弹出的页面中单击“Delete...”，按照提示操作即可。

13.3 运行管理

13.3.1 修改运行配置

步骤1 在MindStudio主界面，在工具栏选择“Edit Configurations...”，如图13-13所示。

图 13-13 配置入口

步骤2 在界面左侧导航栏选择待修改的运行配置信息，并在右侧修改相应的配置信息。

步骤3 单击**Apply**，保存配置，单击**OK**关闭配置界面。

步骤4 单击工具栏的运行按钮运行工程，如图13-14所示。

图 13-14 运行算子工程

----结束

13.3.2 中断运行和重新运行

如果想中断运行UT/ST测试用例，可以单击图13-15所示的 按钮，从而查看运行日志打印。

图 13-15 运行中断按钮（任选其一操作）

如果想重新运行UT/ST测试用例，可以单击 按钮。

13.4 设备管理

本章节讲解如何通过MindStudio界面的Device Manager菜单管理设备信息。

设备管理功能提供了已经连接设备的信息展示，方便使用者对设备进行管理。

- 在MindStudio主界面，单击菜单栏“Ascend > Device Manager”，弹出设备管理窗口。
- 在MindStudio快捷工具栏，单击如下图标，弹出设备管理窗口。

图 13-16 快捷工具栏进入设备管理

如图13-17所示。

图 13-17 Device Manager 窗口

界面参数以及图标解释如表13-2所示。

表 13-2 设备管理界面参数以及图标说明

参数以及图标	说明
Alias	设备的别名，在有多个设备时，方便用户管理设备信息。默认格式为<username>@<ip>:<port>。
Arch	设备所在系统的架构。
Connectivity	MindStudio与设备连接状态： Connected：连接成功。 Disconnected：连接失败。
	编辑设备，单击该按钮可跳转到ssh管理界面，可参考13.1 SSH连接管理进行设备连接信息修改。
	刷新设备，检查设备连接状态，修改了设备信息后，可以通过单击该按钮刷新设备显示信息。

13.5 ADK 管理

13.5.1 概述

ADK Manager为用户提供了在不重装MindStudio的前提下，切换以及更新Ascend-cann-toolkit开发套件包版本的功能。使用该功能之前，请确保已经完成MindStudio以及Ascend-cann-toolkit开发套件包的安装。

ADK管理入口：

- 在MindStudio工程界面菜单栏依次选择“File > Settings... > Appearance&Behavior > System Settings > ADK”，弹出ADK Manager界面。

- 在MindStudio工程界面菜单栏依次选择“Ascend > ADK Manager”，弹出ADK Manager界面。
- 在MindStudio快捷工具栏，单击如下图标进入ADK Manager界面。

图 13-18 快捷工具栏进入 ADK Manager

- 在MindStudio欢迎页面右下角，依次选择“Configure > ADK Manager ”，弹出ADK Manager界面。

4种方式打开的ADK Manager界面如图13-19所示。

图 13-19 ADK Manager 界面

界面参数以及图标解释如表13-3所示。

表 13-3 ADK Manager 界面参数以及图标说明

参数以及图标	说明
ADK Location	ADK包的安装路径，默认为$HOME/Ascend/ascend-toolkit/*{software version}*，通过单击右侧的 可以进行多版本ADK切换。
Component	组件名称。
Version	ADK版本号以及各软件包版本号。
Host OS Arch	Host侧操作系统以及架构。

参数以及图标	说明
Activation	安装的ADK以及软件包是否激活，若显示为Activated，则表示当前安装的ADK版本已经激活，ADK功能可用。
Status	ADK以及软件包的安装状态： ● Installed：ADK或软件包已经安装。 ● Not Installed：ADK或软件包未安装。 只有安装且已经激活的ADK才可用。
	ADK激活按钮。 只有该按钮为可编辑状态才允许该操作。

13.5.2 更新 ADK

通过该章节可以实现不重装MindStudio安装更新ADK版本的功能。

步骤1 请参考“**13.5.1 概述**”进入ADK管理页面，单击页面右侧的“Install ADK”，弹出安装ADK界面。

步骤2 参见**2.3.4 安装MindStudio**进行安装。安装完成后，在设置的安装路径中会生成新版本的ADK信息。

步骤3 MindStudio自动重启后，完成ADK版本的更新。

----结束

13.5.3 切换/激活 ADK

切换 ADK

步骤1 请参考“**13.5.1 概述**”进入ADK管理页面，单击ADK location右侧的 ，系统弹出Toolkit所在路径的页面。如**图13-20**所示。

图 13-20 选择 Toolkit 版本

步骤2 选中需要替换的Toolkit版本，单击“OK”。弹出确认页面，如图13-21所示。

图 13-21 确认页面

- 选择“OK”，ADK切换到新版本，MindStudio重启后该ADK处于激活状态。
  若在切换ADK之前，已经有工程打开，则还会弹出图13-22、图13-23所示界面，用于确认切换后的ADK版本信息。

  图 13-22 ADK 未安装提示信息

图 13-23 确认 ADK 版本信息

- 选择“Cancel”，ADK切换到新版本，但是状态为未激活状态，此时新版本ADK不可用。如果需要激活，请点击“ ”按钮，按照系统提示操作即可。

----结束

激活 ADK

注意

只有“ ”按钮处于非置灰状态时，即当系统存在已经安装但是未激活的ADK时，才可以执行该操作。

请参考“**13.5.1 概述**”进入ADK管理页面，单击ADK对应的“ ”按钮，系统出现提示页面。按照提示操作即可。

13.5.4 配置交叉编译环境

进入ADK管理页面，在ADK管理界面选择“Cross Compiler”页签，系统显示当前激活状态的ADK需要安装的编译环境，如**图13-24**所示。如果Status显示“Not installed”，请参见**2.3.5 配置编译环境**进行手动配置。

图 13-24 配置交叉编译环境

配置完成后，重新启动MindStudio，再次进入ADK管理，可以看到“Status”变成“Installed”。

13.6 日志管理

13.6.1 使用前必读

参考手册

表 13-4 参考手册

产品	手册
Atlas 200 DK	《日志参考》
其他产品	• 《日志参考(推理)》 • 《日志参考(训练)》

工具简介

MindStudio为昇腾AI处理器提供覆盖全系统的日志收集与日志分析解决方案，提升运行时算法问题定位效率。MindStudio提供全系统统一的日志格式，并以图形化的形式提供跨平台日志可视化分析能力及运行时诊断能力，提升日志分析系统的易用性。

通过Log工具，您可以进行以下操作：

- 日志管理
- 设置日志级别

在MindStudio窗口底部单击“+Log”标签，显示“Log”窗口，您可以通过该窗口查看日志信息，如图13-25所示。

图 13-25 log 界面

说明

- localhost默认始终置顶存在，用于查看本地的系统运行日志。
- 在Linux系统下使用Log工具打开文件夹和文件功能，需要使用nautilus、gedit软件；如果系统未安装此软件，请使用MindStudio安装用户执行以下命令进行安装。
  - Ubuntu18.04-x86/aarch系统：**sudo apt-get install nautilus gedit**。
  - EulerOS2.8-aarch系统：**sudo yum install nautilus gedit**。

13.6.2 日志管理

在“Log”窗口的System Log页签里，您可以查看系统运行日志，操作步骤如下：

说明

- MindStudio不支持通过界面方式删除设备上的System Log日志。如需对日志进行管理：
  - RC场景：请使用root用户登录到设备侧，对/var/log/npu/slog目录下的日志进行管理操作。
  - EP场景：请使用通过SSH连接到的用户对$HOME/ascend/log/目录下的日志进行管理操作。
- 进行系统日志查看前，需要已添加Device设备，才能获取到Device设备的相关日志。

步骤1 在MindStudio界面的“Log”窗口，单击“System Log”页签，在显示的设备列表中，选中某个设备，展开该设备的日志列表。

图 13-26 日志列表（RC 模式下）

图 13-27 日志列表（EP 模式下）

步骤2 双击日志列表中某日志文件，展示日志内容。

说明

本处最多展示该日志文件中最新的5000条日志，且最多展示1M的日志内容。

图 13-28 System log 页签（RC 模式下）

图 13-29 System log 页签（EP 模式下）

步骤3 在System Log页签，您可以进行如下操作。

- **单击** 图标： 提供查询功能，按关键字搜索日志内容。
- **单击** 图标：清除显示的日志。
- **单击** Open Log File 图标： 打开日志文件，展示该日志文件的全部日志内容。
- **单击** Open Log Folder 图标：打开日志文件下载到的IDE所在机器的本地存储路径。

----结束

13.6.3 设置日志级别

Log工具提供界面方式修改系统的日志级别，设置日志级别方法：

步骤1 在MindStudio界面的“Log”窗口，单击“System Log”页签，在显示的Device设备列表中，选中某个设备，单击右键，选择“Set Log Level”。如图13-30所示。

图 13-30 选择 Set Log Level

步骤2 在弹出“Set Log Level”窗口，您可以设置日志级别。如图13-31所示。

说明

- 日志级别说明请参考表13-4中的《日志参考》。
- 执行修改日志级别后，针对PCIe场景，如果重启昇腾AI处理器，Device侧日志级别将恢复为系统默认级别。

图 13-31 Set Log Level 窗口

步骤3 设置日志级别后，单击“OK”。

----结束

13.7 常用快捷键

本节介绍IntelliJ IDEA的常用快捷键，了解更多内容请参考相关官方文档。

表 13-5 常用快捷键

快捷键	说明
Alt + 左方向键	切换当前已打开的窗口中的子视图，比如Debug窗口中有Output、Debugger等子视图，用此快捷键就可以在子视图中切换（必备）
Alt + 右方向键	按切换当前已打开的窗口中的子视图，比如Debug窗口中有Output、Debugger等子视图，用此快捷键就可以在子视图中切换（必备）
Alt + 前方向键	当前光标跳转到当前文件的前一个方法名位置（必备）
Alt + 后方向键	当前光标跳转到当前文件的后一个方法名位置（必备）
Ctrl + =	展开代码
Ctrl + -	折叠代码
Ctrl + /	注释光标所在行代码，会根据当前不同文件类型使用不同的注释符号（必备）
Ctrl + F	在当前文件进行文本查找（必备）
Ctrl + R	在当前文件进行文本替换（必备）
Ctrl + Z	撤销（必备）
Ctrl + Y	删除光标所在行 或 删除选中的行（必备）
Ctrl + X	剪切光标所在行 或 剪切选择内容
Ctrl + C	复制光标所在行 或 复制选择内容

快捷键	说明
Ctrl + D	复制光标所在行 或 复制选择内容，并把复制内容插入光标位置下面（必备）
Ctrl + W	递进式选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展选中范围（必备）
Ctrl + E	显示最近打开的文件记录列表（必备）
Ctrl + N	根据输入的 **类名** 查找类文件（必备）
Ctrl + G	在当前文件跳转到指定行处
Ctrl + Q	光标所在的变量 / 类名 / 方法名等上面，显示文档内容
Ctrl + Alt + L	格式化代码，可以对当前文件和整个包目录使用（必备）
Ctrl + Alt + O	优化导入的类，可以对当前文件和整个包目录使用（必备）
Ctrl + Shift + F	根据输入内容查找整个项目 或 指定目录内文件（必备）
Ctrl + Shift + R	根据输入内容替换对应内容，范围为整个项目 或 指定目录内文件（必备）

13.8 常用原生功能

本节介绍IntelliJ IDEA的常用功能，了解更多内容请参考相关官方文档。

折叠目录

图13-32中的参数去勾选表示不折叠目录，勾选表示折叠目录。

图 13-32 配置

图13-33为不折叠目录和折叠目录的比较。

图 13-33 目录比较

Project
custom ~/MindStudio/samples/demo_projects/cu
.idea
build
cmake
framework
op_proto
scripts
tbe
impl
op_info_cfg
ai_core
ascend310
ascend610
ascend710
ascend910
hi3796cv300es
CMakeLists.txt
Project
custom ~/MindStudio/samples/demo_projects/cu
.idea
build
cmake
framework
op_proto
scripts
tbe
impl
op_info_cfg.ai_core
ascend310
ascend610
ascend710
ascend910
hi3796cv300es
CMakeLists.txt

14 MindStudio 管理

14.1 查询MindStudio版本
14.2 启动MindStudio
14.3 停止MindStudio
14.4 更新MindStudio
14.5 卸载MindStudio
14.6 FAQ

14.1 查询 MindStudio 版本

安装完成后您可以通过如下方式查询所安装MindStudio版本是否正确：

- MindStudio安装完成后，在欢迎界面中，单击右下角的“Get Help > About”查看所安装的版本信息，如图14-1所示：

图 14-1 查询版本号

- 进入MindStudio工程页面，单击菜单栏的“Help > About”弹出版本信息，查看版本信息是否正确。

说明

图14-1中的版本号只是样例，请以实际安装的版本为准。

14.2 启动 MindStudio

如果用户关闭了MindStudio，想要重新启动，则可以参见如下方式操作：

使用MindStudio的安装用户，进入软件包解压后的“MindStudio/bin”目录，执行如下命令启动MindStudio：

```
./MindStudio.sh
```

启动MindStudio后，查询进程命令为：

```
ps -ef|grep idea
```

14.3 停止 MindStudio

如果用户想要关闭MindStudio，在MindStudio运行界面直接单击窗口右上角的退出即可。

14.4 更新 MindStudio

手动更新

建议使用最新版本的MindStudio，以便获得最新功能。如果尚未安装MindStudio，请参见2.3.4 安装MindStudio安装。

MindStudio为绿色版软件，如果用户想更新版本，请参见14.5 卸载MindStudio章节卸载已安装的MindStudio， 然后参见2.3.4 安装MindStudio重新安装新版本MindStudio。

通过 msInstaller 更新

请参见《CANN 软件安装指南 (开发场景，通过msInstaller工具)》中的升级章节。

14.5 卸载 MindStudio

如果用户不再使用MindStudio与Ascend-cann-toolkit开发套件包功能，则可以参见该章节将其卸载。如果用户只卸载Toolkit，则请慎重，可能会造成MindStudio其他功能不可用。如果用户想更换Ascend-cann-toolkit开发套件包版本，则可以参见13.5 ADK管理章节安装新版本Ascend-cann-toolkit开发套件包。

手动卸载 MindStudio

用MindStudio的安装用户，在linux系统的$HOME目录下删除MindStudio安装包解压后的目录：MindStudio。

删除命令为：

```
rm -rf MindStudio
```

- 卸载MindStudio不会删除如下目录中的数据：
  - $HOME/.cache/Huawei/MindStudioMS-*{version}*：MindStudio安装以及启动后的运行日志等信息。
  - $HOME/.config/Huawei/MindStudioMS-*{version}*：用户配置数据。
  - $HOME/.mindstudio：包括Toolkit安装路径（mindstudio.properties文件）、MindStudio隐私声明界面用户所做的操作信息。
  - AscendProjects：用户创建的工程。
- 如果用户想要完全恢复MindStudio初始配置，则请使用MindStudio安装用户进入$HOME目录分别执行如下命令，删除用户配置数据：

```
rm -rf .cache/Huawei/MindStudioMS-x.x/
rm -rf .config/Huawei/MindStudioMS-x.x/
rm -rf .mindstudio/
```

手动卸载 Ascend-cann-toolkit 开发套件包

请参见《**CANN 软件安装指南 (开发&运行场景，通过命令行方式)**》中的卸载章节。

通过 msInstaller 卸载

请参见《**CANN 软件安装指南 (开发场景，通过msInstaller工具)**》中的卸载章节。

14.6 FAQ

14.6.1 使用 pip3 install 软件时提示" Could not find a version that satisfies the requirement xxx"

问题描述

安装依赖时，使用**pip3 install xxx**命令安装相关软件时提示无法连接网络，且提示“Could not find a version that satisfies the requirement xxx"，使用**apt-get update**命令检查源可用。提示信息如下所示。

图 14-2 使用 pip3 安装软件提示信息

```
ascend@dggphicprd32833:~$ pip3 install numpy --user
Collecting numpy
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
7f5f725b9320>: Failed to establish a new connection: [Errno 101] Network is unreachable',)': /simple/numpy/
  Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
7f5f725cd908>: Failed to establish a new connection: [Errno 101] Network is unreachable',)': /simple/numpy/
  Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
7f5f725cd9e8>: Failed to establish a new connection: [Errno 101] Network is unreachable',)': /simple/numpy/
  Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
7f5f725cdac8>: Failed to establish a new connection: [Errno 101] Network is unreachable',)': /simple/numpy/
  Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
7f5f725cdb38>: Failed to establish a new connection: [Errno 101] Network is unreachable',)': /simple/numpy/
  Could not find a version that satisfies the requirement numpy (from versions: )
```

可能原因

没有配置pip源。

解决方法

配置pip源，配置方法如下：

步骤1 使用MindStudio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
```

在.pip目录下创建pip.conf 文件，命令为：

```
touch pip.conf
```

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：

```
[install]
#可信主机，请根据实际情况进行替换。
trusted-host=repo.huaweicloud.com
[global]
#可用的源，请根据实际情况进行替换。
index-url=https://repo.huaweicloud.com/repository/pypi/simple
```

步骤3 执行**:wq!**命令保存文件。

----结束

15 附录

15.1 软件包说明
15.2 安装依赖
15.3 算子规格说明
15.4 特性配置
15.5 使用AIPP色域转换模型时如何判断视频流的格式标准
15.6 配置系统网络代理
15.7 安全加固建议
15.8 FAQ
15.9 修订记录

15.1 软件包说明

MindStudio安装包解压后包的详细内容请参见**表15-1**。

表 15-1 解压后软件包说明

安装包	包内容	使用场景
MindStudio_{*version*}_linux.tar.gz	MindStudio： ● bin：MindStudio的执行目录及依赖的二进制文件 ● build.txt：安装包构建信息 ● config：MindStudio配置文件 ● Install-Linux-tar.txt：MindStudio安装说明 ● jbr：64位系统的依赖库以及文件 ● lib：JAVA依赖库 ● license：所用的第三方依赖专利文档 ● LICENSE.txt：Apache Licence说明文档 ● NOTICE.txt：注意事项 ● plugins：MindStudio所用到的基本插件及JAVA库 ● product-info.json：MindStudio版本号以及执行文档路径 ● redist：Java辅助注解 ● samples：算子样例工程模板文件以及典型网络模型的模型文件 ● tools：工具库	MindStudio工具安装包

15.2 安装依赖

15.2.1 安装依赖(Ubuntu-x86_64)

步骤1 使用**su - *username***命令切换到MindStudio安装用户。

步骤2 检查系统是否安装依赖软件。

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、g++、make以及python依赖软件等。

```
gcc --version
g++ --version
make --version
cmake --version
unzip --version
dpkg -l zlib1g| grep zlib1g| grep ii
dpkg -l zlib1g-dev| grep zlib1g-dev| grep ii
dpkg -l libbz2-dev| grep libbz2-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 libxslt1-dev| grep libxslt1-dev| grep ii
dpkg -l libffi-dev| grep libffi-dev| grep ii
```

```
dpkg -l libncurses5-dev| grep libncurses5-dev| grep ii
dpkg -l libncursesw5-dev| grep libncursesw5-dev| grep ii
dpkg -l gnome-keyring| grep gnome-keyring| grep ii
dpkg -l pciutils| grep pciutils| grep ii
dpkg -l net-tools| grep net-tools| grep ii
```

若分别返回如下信息则说明已经安装。

```
g++ (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
GNU Make 4.1
cmake version  3.10.2
UnZip 6.00 of 20 April 2009, by Debian. Original by Info-ZIP.
zlib1g:amd64   1:1.2.11.dfsg-0ubuntu2 amd64        compression library - runtime
zlib1g-dev:amd64 1:1.2.8.dfsg-2ubuntu4.1 amd64        compression library - development
libbz2-dev:amd64 1.0.6-8ubuntu0.2 amd64        high-quality block-sorting file compressor library -
development
libsqlite3-dev:amd64 3.11.0-1ubuntu1.2 amd64        SQLite 3 development files
openssl        1.1.1-1ubuntu2.1~18.04.5 amd64        Secure Sockets Layer toolkit - cryptographic utility
libssl-dev:amd64 1.0.2g-1ubuntu4.15 amd64        Secure Sockets Layer toolkit - development files
libxslt1-dev:amd64 1.1.28-2.1ubuntu0.3 amd64        XSLT 1.0 processing library - development kit
libffi-dev:amd64 3.2.1-4      amd64        Foreign Function Interface library (development files)
libncurses5-dev:amd64 6.1-1ubuntu1.18.04 amd64        developer's libraries for ncurses
libncursesw5-dev:amd64
gnome-keyring  3.28.0.2-1ubuntu1.18.04.1 amd64        GNOME keyring services (daemon and tools)
pciutils       1:3.5.2-1ubuntu1.1 amd64        Linux PCI Utilities
net-tools      1.60+git20161116.90da8a0-1ubuntu1 amd64        NET-3 networking toolkit
```

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）：

```
sudo apt-get install -y gcc g++ make cmake unzip zlib1g zlib1g-dev libbz2-dev libsqlite3-dev openssl
libssl-dev libxslt1-dev libffi-dev libncurses5-dev libncursesw5-dev gnome-keyring pciutils net-tools
```

2. 检查系统是否安装python开发环境。

   ATC安装依赖python3环境，分别使用命令**python3.7.5 --version**、**pip3.7.5 --version**检查是否已经安装，如果分别返回如下信息则说明已经安装。

```
Python 3.7.5
pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

   否则请根据如下方式安装python3.7.5。

   a. 使用wget下载python3.7.5源码包，可以下载到MindStudio安装服务器任意目录，命令为：

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

   b. 进入下载后的目录，解压源码包，命令为：

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

   c. 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
sudo mkdir /usr/local/python3.7.5
./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。

   d. 执行如下命令复制libpython3.7m.so.1.0动态库：

      i. 将编译后的文件libpython3.7m.so.1.0复制到/usr/lib64。

```
sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib64
```

      如果出现如下显示，则输入y，表示覆盖系统自带的libpython3.7m.so.1.0文件。

```
cp: overwrite 'libpython3.7m.so.1.0'?
```

如果环境上没有/usr/lib64，则复制到/usr/lib目录：

```
sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib
```

libpython3.7m.so.1.0文件所在路径请根据实际情况进行替换。

e. 执行如下命令设置软链接：

```
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5
```

f. 设置python3.7.5环境变量。

i. 以安装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如下内容。

```
#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
```

ii. 执行**:wq!**命令保存文件并退出。

iii. 执行**source ~/.bashrc**命令使其立即生效。

g. 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。

```
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7  --version
```

3. 检查是否安装MindStudio显示运行依赖的库文件。

MindStudio依赖xterm显示依赖运行的库文件，使用命令**dpkg -l xterm| grep xterm| grep ii**检查是否已经安装，若返回如下信息则说明已经安装。

```
xterm          322-1ubuntu1 amd64        X terminal emulator
```

否则请执行如下安装命令：

```
sudo apt-get install -y xterm
```

4. 检查是否安装MindStudio显示html报告依赖的软件。

MindStudio依赖firefox、xdg-utils显示html报告，分别使用**firefox --version**、**dpkg -l xdg-utils| grep xdg-utils| grep ii**命令检查是否已经安装，若分别返回如下信息则说明已经安装。

```
Mozilla Firefox 71.0
xdg-utils      1.1.1-1ubuntu1.16.04.3 all          desktop integration utilities from freedesktop.org
```

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）：

```
sudo apt-get install -y firefox xdg-utils
```

5. 检查是否安装字体软件。

MindStudio中查看联机帮助需要借助相关字体软件，分别使用如下命令检查是否已经安装：

```
dpkg -l fonts-droid-fallback| grep fonts-droid-fallback| grep ii
dpkg -l fonts-wqy-zenhei| grep fonts-wqy-zenhei| grep ii
dpkg -l fonts-wqy-microhei| grep fonts-wqy-microhei| grep ii
dpkg -l fonts-arphic-ukai| grep fonts-arphic-ukai| grep ii
dpkg -l fonts-arphic-uming| grep fonts-arphic-uming| grep ii
```

若分别返回如下信息则说明已经安装。

```
fonts-droid-fallback 1:6.0.1r16-1 all          handheld device font with extensive style and language support (fallback)
fonts-wqy-zenhei 0.9.45-6ubuntu1 all          "WenQuanYi Zen Hei" A Hei-Ti Style (sans-serif) Chinese font
fonts-wqy-microhei 0.2.0-beta-2 all          Sans-serif style CJK font derived from Droid
fonts-arphic-ukai 0.2.20080216.2-4ubuntu2 all          "AR PL UKai" Chinese Unicode TrueType font collection Kaiti style
```

```
fonts-arphic-uming 0.2.20080216.2-7ubuntu2 all "AR PL UMing" Chinese Unicode TrueType font collection Mingti style
```

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）：

```
sudo apt-get install -y fonts-droid-fallback fonts-wqy-zenhei fonts-wqy-microhei fonts-arphic-ukai fonts-arphic-uming
```

步骤3 安装 Python3依赖。

安装TBE依赖软件：numpy，decorator；TIK依赖软件：sympy，cffi，gnureadline；utst工具依赖软件：coverage，精度比对依赖protobuf、scipy；profiling工具依赖：protobuf、grpcio、grpcio-tools、requests。若用户使用算子ST用例从模型获取shape等功能，则需要安装tensorflow，否则不用安装。

安装前请先使用**pip3.7.5 list**命令检查是否安装相关依赖，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）：

```
pip3.7.5 install attrs --user
pip3.7.5 install psutil --user
pip3.7.5 install decorator --user
pip3.7.5 install numpy --user
pip3.7.5 install protobuf==3.11.3 --user
pip3.7.5 install scipy --user
pip3.7.5 install sympy --user
pip3.7.5 install cffi --user
pip3.7.5 install gnureadline --user
pip3.7.5 install coverage --user
pip3.7.5 install pylint --user
pip3.7.5 install matplotlib --user
pip3.7.5 install grpcio --user
pip3.7.5 install grpcio-tools --user
pip3.7.5 install requests --user
pip3.7.5 install xlrd==1.2.0 --user
pip3.7.5 install absl-py --user
pip3.7.5 install pandas --user
pip3.7.5 install tensorflow==1.15.0 --user
pip3.7.5 install pytest --user
```

如果执行上述命令时无法连接网络，且提示“Could not find a version that satisfies the requirement xxx”，请参见**安装-FAQ**解决。

步骤4 安装JDK。

安装JDK之前，先用如下命令检查系统是否已经存在JAVA：

```
which java
```

若返回相关路径，则说明已经存在，否则执行如下命令安装JDK。

```
sudo apt-get install -y openjdk-8-jdk
```

1. 配置JAVA_HOME环境变量，MindStudio的安装及运行都依赖该环境变量，设置方法如下：
   a. 在任何目录下执行如下命令，打开**.bashrc**文件。
      ```
      vi ~/.bashrc
      ```
   b. 在文件的最后一行后面添加如下内容。
      ```
      export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
      export PATH=$JAVA_HOME/bin:$PATH
      ```

      说明

      “JAVA_HOME”为JDK的安装目录，若用户已经配置了JDK，请根据安装目录的实际情况进行修改。若根据上述步骤安装的JDK，则安装目录不用修改。

   c. 执行如下命令保存文件并退出。
      ```
      :wq!
      ```

d. 执行如下命令使环境变量生效。
```
source ~/.bashrc
```
e. 执行如下命令检查环境变量设置：
```
echo $JAVA_HOME
```
回显信息如下：
```
/usr/lib/jvm/java-8-openjdk-amd64
```
f. 执行如下命令检查JDK安装。
```
which jconsole
```
如果输出如下回显信息表示安装成功，否则表示JDK安装失败。
```
/usr/lib/jvm/java-8-openjdk-amd64/bin/jconsole
```

步骤5 返回**2.3.4 安装MindStudio**。

----结束

15.2.2 安装依赖(Ubuntu-aarch64)

步骤1 使用**su - *username***命令切换到MindStudio安装用户。

步骤2 检查系统是否安装依赖软件。

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、g++、make等依赖软件。
```
gcc --version
g++ --version
make --version
cmake --version
unzip --version
dpkg -l zlib1g| grep zlib1g| grep ii
dpkg -l zlib1g-dev| grep zlib1g-dev| grep ii
dpkg -l libbz2-dev| grep libbz2-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 libxslt1-dev| grep libxslt1-dev| grep ii
dpkg -l libffi-dev| grep libffi-dev| grep ii
dpkg -l libncurses5-dev| grep libncurses5-dev| grep ii
dpkg -l gnome-keyring| grep gnome-keyring| 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
dpkg -l libopenblas-dev| grep libopenblas-dev| grep ii
dpkg -l build-essential | grep build-essential  | grep ii
dpkg -l libgdbm-dev | grep libgdbm-dev  | grep ii
dpkg -l libncursesw5-dev | grep libncursesw5-dev  | grep ii
dpkg -l pkg-config | grep pkg-config  | grep ii
dpkg -l libglib2.0-dev | grep libglib2.0-dev  | grep ii
dpkg -l libdbus-glib-1-dev | grep libdbus-glib-1-dev  | grep ii
dpkg -l gdb | grep gdb  | grep ii
dpkg -l liblapack-dev  | grep liblapack-dev  | grep ii
dpkg -l m4 | grep m4  | grep ii
dpkg -l openjfx | grep openjfx  | grep ii
dpkg -l libfreetype6-dev | grep libfreetype6-dev  | grep ii
dpkg -l libqhull-dev | grep libqhull-dev  | grep ii
dpkg -l openssh-server| grep openssh-server| grep ii
```
   若分别返回如下信息则说明已经安装。
```
gcc (Ubuntu/Linaro 7.5.0-3ubuntu1~18.04) 7.5.0
g++ (Ubuntu/Linaro 7.5.0-3ubuntu1~18.04) 7.5.0
GNU Make 4.1
cmake version  3.10.2
UnZip 6.00 of 20 April 2009, by Debian. Original by Info-ZIP.
ii  zlib1g:arm64   1:1.2.11.dfsg-0ubuntu2 arm64        compression library - runtime
```

```
ii  zlib1g-dev:arm64 1:1.2.11.dfsg-0ubuntu2 arm64        compression library - development
ii  libbz2-dev:arm64 1.0.6-8.1ubuntu0.2 arm64        high-quality block-sorting file compressor library -
development
ii  libsqlite3-dev:arm64 3.22.0-1ubuntu0.4 arm64        SQLite 3 development files
ii  openssl        1.1.1-1ubuntu2.1~18.04.7 arm64        Secure Sockets Layer toolkit - cryptographic
utility
ii  libssl-dev:arm64 1.1.1-1ubuntu2.1~18.04.7 arm64        Secure Sockets Layer toolkit - development
files
dpkg-query: no packages found matching libxslt1-dev
ii  libffi-dev:arm64 3.2.1-8      arm64        Foreign Function Interface library (development files)
ii  libncurses5-dev:arm64 6.1-1ubuntu1.18.04 arm64        developer's libraries for ncurses
ii  gnome-keyring  3.28.0.2-1ubuntu1.18.04.1 arm64        GNOME keyring services (daemon and tools)
dpkg-query: no packages found matching pciutils
dpkg-query: no packages found matching net-tools
ii  gfortran       4:7.4.0-1ubuntu2.3 arm64        GNU Fortran 95 compiler
ii  libopenblas-dev:arm64 0.2.20+ds-4  arm64        Optimized BLAS (linear algebra) library
(development files)
ii  build-essential 12.4ubuntu1  arm64        Informational list of build-essential packages
ii  libgdbm-dev:arm64 1.14.1-6     arm64        GNU dbm database routines (development files)
ii  libncursesw5-dev:arm64 6.1-1ubuntu1.18.04 arm64        developer's libraries for ncursesw
ii  pkg-config     0.29.1-0ubuntu2 arm64        manage compile and link flags for libraries
ii  libglib2.0-dev:arm64 2.56.4-0ubuntu0.18.04.6 arm64        Development files for the GLib library
ii  libdbus-glib-1-dev 0.110-2      arm64        deprecated library for D-Bus IPC (development files)
ii  gdb            8.1.1-0ubuntu1 arm64        GNU Debugger
ii  liblapack-dev:arm64 3.7.1-4ubuntu1 arm64        Library of linear algebra routines 3 - static version
ii  m4             1.4.18-1     arm64        macro processing language
ii  openjfx        11.0.2+1-1~18.04.2 arm64        JavaFX/OpenJFX - Rich client application platform for
Java
ii  libfreetype6-dev:arm64 2.8.1-2ubuntu2.1 arm64        FreeType 2 font engine, development files
ii  libqhull-dev:arm64 2015.2-4     arm64        calculate convex hulls and related structures
(development files)
ii  openssh-server 1:7.6p1-4ubuntu0.3 arm64        secure shell (SSH) server, for secure access from
remote machines
```

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）：

```
sudo apt-get install -y gcc g++ make cmake unzip zlib1g zlib1g-dev libbz2-dev libsqlite3-dev openssl
libssl-dev libxslt1-dev libffi-dev libncurses5-dev gnome-keyring pciutils net-tools libblas-dev gfortran
libblas3 libopenblas-dev build-essential libgdbm-dev libncursesw5-dev pkg-config libglib2.0-dev
libdbus-glib-1-dev gdb liblapack-dev m4 openjfx libfreetype6-dev libqhull-dev openssh-server
```

2. 检查系统是否安装python开发环境。

   ATC安装依赖python3环境，分别使用命令**python3.7.5 --version**、**python3.7 --version**、**pip3.7.5 --version**、**pip3 --version**检查是否已经安装，如果分别返回如下信息则说明已经安装。

```
Python 3.7.5
pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

   否则请根据如下方式安装python3.7.5。

   a. 使用wget下载python3.7.5源码包，可以下载到MindStudio安装服务器任意目录，命令为：

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

   b. 进入下载后的目录，解压源码包，命令为：

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

   c. 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
sudo mkdir /usr/local/python3.7.5
./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

d. 执行如下命令复制libpython3.7m.so.1.0动态库：

   i. 将编译后的文件libpython3.7m.so.1.0复制到/usr/lib64。
   ```
   sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib64
   ```
   如果出现如下显示，则输入y，表示覆盖系统自带的libpython3.7m.so.1.0文件。
   ```
   cp: overwrite 'libpython3.7m.so.1.0'?
   ```
   如果环境上没有/usr/lib64，则复制到/usr/lib目录：
   ```
   sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib
   ```
   libpython3.7m.so.1.0文件所在路径请根据实际情况进行替换。

e. 执行如下命令设置软链接：
```
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5
```

f. 设置python3.7.5环境变量。

   i. 以安装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如下内容。
   ```
   #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
   ```
   ii. 执行**:wq!**命令保存文件并退出。

   iii. 执行**source ~/.bashrc**命令使其立即生效。

g. 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。
```
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7  --version
```

3. 检查是否安装MindStudio显示运行依赖的库文件。

   MindStudio依赖xterm显示依赖运行的库文件，使用命令**dpkg -l xterm| grep xterm| grep ii**检查是否已经安装，若返回如下信息则说明已经安装。
   ```
   xterm        322-1ubuntu1 amd64        X terminal emulator
   ```
   否则请执行如下安装命令：
   ```
   sudo apt-get install -y xterm
   ```

4. 检查是否安装MindStudio显示html报告依赖的软件。

   MindStudio依赖firefox、xdg-utils显示html报告，分别使用**firefox --version**、**dpkg -l xdg-utils| grep xdg-utils| grep ii**命令检查是否已经安装，若分别返回如下信息则说明已经安装。
   ```
   Mozilla Firefox 71.0
   xdg-utils    1.1.1-1ubuntu1.16.04.3 all        desktop integration utilities from freedesktop.org
   ```
   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）：
   ```
   sudo apt-get install -y firefox xdg-utils
   ```

5. 检查是否安装字体软件。

   MindStudio中查看联机帮助需要借助相关字体软件，分别使用如下命令检查是否已经安装：

```
dpkg -l fonts-droid-fallback| grep fonts-droid-fallback| grep ii
dpkg -l fonts-wqy-zenhei| grep fonts-wqy-zenhei| grep ii
dpkg -l fonts-wqy-microhei| grep fonts-wqy-microhei| grep ii
dpkg -l fonts-arphic-ukai| grep fonts-arphic-ukai| grep ii
dpkg -l fonts-arphic-uming| grep fonts-arphic-uming| grep ii
```

若分别返回如下信息则说明已经安装。

```
fonts-droid-fallback 1:6.0.1r16-1 all          handheld device font with extensive style and language support (fallback)
fonts-wqy-zenhei 0.9.45-6ubuntu1 all          "WenQuanYi Zen Hei" A Hei-Ti Style (sans-serif) Chinese font
fonts-wqy-microhei 0.2.0-beta-2 all          Sans-serif style CJK font derived from Droid
fonts-arphic-ukai 0.2.20080216.2-4ubuntu2 all          "AR PL UKai" Chinese Unicode TrueType font collection Kaiti style
fonts-arphic-uming 0.2.20080216.2-7ubuntu2 all          "AR PL UMing" Chinese Unicode TrueType font collection Mingti style
```

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）：

```
sudo apt-get install -y fonts-droid-fallback fonts-wqy-zenhei fonts-wqy-microhei fonts-arphic-ukai fonts-arphic-uming
```

步骤3 编译安装matplotlib。

1. 使用wget下载matplotlib-3.3.3源码包，可以下载到MindStudio安装服务器任意目录，命令为：

```
wget https://files.pythonhosted.org/packages/7b/b3/7c48f648bf83f39d4385e0169d1b68218b838e185047f7f613b1cfc57947/matplotlib-3.3.3.tar.gz
```

2. 进入下载后的目录，解压源码包，命令为：

```
tar -zxvf matplotlib-3.3.3.tar.gz
```

3. 进入解压后的文件夹，执行配置、安装命令：

```
cd matplotlib-3.3.3
echo "[libs]" >> ./setup.cfg
echo "system_freetype=true" >> ./setup.cfg
echo "system_qhull=true" >> ./setup.cfg
python3.7.5 -m pip install .
```

步骤4 安装 Python3依赖。

安装TBE依赖软件：numpy，decorator；TIK依赖软件：sympy，cffi，gnureadline；utst工具依赖软件：coverage，精度比对依赖protobuf、scipy；profiling工具依赖：protobuf、grpcio、grpcio-tools、requests。若用户使用算子ST用例从模型获取shape等功能，则需要安装tensorflow，否则不用安装。

安装前请先使用**pip3.7.5 list**命令检查是否安装相关依赖，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）：

```
pip3.7.5 install attrs --user
pip3.7.5 install psutil --user
pip3.7.5 install decorator --user
pip3.7.5 install numpy --user
pip3.7.5 install protobuf==3.11.3 --user
pip3.7.5 install scipy --user
pip3.7.5 install sympy --user
pip3.7.5 install cffi --user
pip3.7.5 install gnureadline --user
pip3.7.5 install coverage --user
pip3.7.5 install pylint --user
pip3.7.5 install grpcio --user
pip3.7.5 install grpcio-tools --user
pip3.7.5 install requests --user
pip3.7.5 install xlrd==1.2.0 --user
pip3.7.5 install absl-py --user
pip3.7.5 install pandas --user
pip3.7.5 install pytest --user
```

如果执行上述命令时无法连接网络，且提示“Could not find a version that satisfies the requirement xxx”，请参见**安装-FAQ**解决。

步骤5 安装JDK。

1. 安装JDK之前，先用如下命令检查系统是否已经存在JAVA：
```
which java
```
若返回相关路径，则说明已经存在，否则执行如下命令安装JDK。
```
sudo apt-get install -y openjdk-8-jdk
```
2. 配置JAVA_HOME环境变量，MindStudio的安装及运行都依赖该环境变量，设置方法如下：
   a. 在任何目录下执行如下命令，打开**.bashrc**文件。
   ```
   vi ~/.bashrc
   ```
   b. 在文件的最后一行后面添加如下内容。
   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-arm64
   export PATH=$JAVA_HOME/bin:$PATH
   ```
   说明

   “JAVA_HOME”为JDK的安装目录，若用户已经配置了JDK，请根据安装目录的实际情况进行修改。若根据上述步骤安装的JDK，则安装目录不用修改。

   c. 执行如下命令保存文件并退出。
   ```
   :wq!
   ```
   d. 执行如下命令使环境变量生效。
   ```
   source ~/.bashrc
   ```
   e. 执行如下命令检查环境变量设置：
   ```
   echo $JAVA_HOME
   ```
   回显信息如下：
   ```
   /usr/lib/jvm/java-8-openjdk-arm64
   ```
   f. 执行如下命令检查JDK安装。
   ```
   which jconsole
   ```
   如果输出如下回显信息表示安装成功，否则表示JDK安装失败。
   ```
   /usr/lib/jvm/java-8-openjdk-arm64/bin/jconsole
   ```

步骤6 （可选）安装Tensorflow 1.15.0。具体操作请参见**安装Tensorflow 1.15.0**。

步骤7 返回**2.3.4 安装MindStudio**。

----结束

安装 Tensorflow 1.15.0

需要安装Tensorflow 1.15.0才可以进行算子开发验证、训练业务开发。

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。

1. 编译安装HDF5。
   a. 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
   ```
   b. 进入下载后的目录，解压源码包，命令为：
   ```
   tar -zxvf hdf5-1.10.5.tar.gz
   ```
   c. 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
   ```

2. 配置环境变量并建立动态链接库软连接。
   a. 配置环境变量。
   ```
   export CPATH="/usr/include/hdf5/include/:/usr/include/hdf5/lib/"
   ```
   b. 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
   ```
3. 安装h5py。
   a. root用户下安装h5py依赖包命令如下。
   ```
   pip3.7 install Cython
   ```
   b. 安装h5py命令如下。
   ```
   pip3.7 install h5py==2.8.0
   ```
4. 安装grpcio。

   root用户下安装grpcio命令如下。
   ```
   pip3.7 install grpcio==1.32.0
   ```

安装步骤

1. 下载已经编译好的Tensorflow1.15软件包：**tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl**
2. 安装编译好的Tensorflow。

   进入软件包所在目录，root用户执行如下命令安装Tensorflow1.15：
   ```
   pip3.7 install tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl
   ```
   非root用户执行如下命令：
   ```
   pip3.7 install tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl --user
   ```

15.2.3 安装依赖(EulerOS-aarch64)

步骤1 使用**su - *username***命令切换到MindStudio安装用户。

步骤2 检查系统是否安装依赖软件。

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、g++、make等依赖软件。
   ```
   gcc --version
   g++ --version
   make --version
   cmake --version
   rpm -qa | grep automake
   rpm -qa | grep kernel-devel
   rpm -qa | grep zlib-devel
   rpm -qa | grep bzip2-devel
   rpm -qa | grep sqlite-devel
   rpm -qa | grep openssl-devel
   rpm -qa | grep libxslt-devel
   rpm -qa | grep libffi-devel
   rpm -qa | grep gnome-keyring
   rpm -qa | grep glib2
   rpm -qa | grep dbus-glib-devel
   rpm -qa | grep xterm
   rpm -qa | grep openssh-clients
   rpm -qa | grep openssh-server
   rpm -qa | grep ncurses-devel
   rpm -qa | grep readline-devel
   rpm -qa | grep gdb-gdbserve
   rpm -qa | grep atlas
   rpm -qa | grep atlas-devel
   rpm -qa | grep lapack
   ```

```
rpm -qa | grep lapack-devel
rpm -qa | grep gcc-gfortran
rpm -qa | grep initscripts
rpm -qa | grep freetype-devel
rpm -qa | grep libpng-deve
rpm -qa | grep xdg-utils
```

若分别返回如下信息则说明已经安装。

```
gcc (GCC) 7.3.0
g++ (GCC) 7.3.0
GNU Make 4.2.1
cmake version 3.12.1
automake-1.16.1-5.eulerosv2r8.noarch
kernel-devel-4.19.36-vhulk1907.1.0.h821.eulerosv2r8.aarch64
zlib-devel-1.2.11-14.h1.eulerosv2r8.aarch64
bzip2-devel-1.0.6-29.h2.eulerosv2r8.aarch64
sqlite-devel-3.24.0-2.h19.eulerosv2r8.aarch64
openssl-devel-1.1.1-3.h11.eulerosv2r8.aarch64
libxslt-devel-1.1.32-3.h6.eulerosv2r8.aarch64
libffi-devel-3.1-18.h3.eulerosv2r8.aarch64
gnome-keyring-3.28.2-2.eulerosv2r8.aarch64
glib2-devel-2.58.1-1.h4.eulerosv2r8.aarch64
glib2-2.58.1-1.h4.eulerosv2r8.aarch64
dbus-glib-devel-0.110-3.h3.eulerosv2r8.aarch64
xterm-334-1.eulerosv2r8.aarch64
xterm-resize-334-1.eulerosv2r8.aarch64
openssh-clients-7.8p1-3.h39.eulerosv2r8.aarch64
openssh-server-7.8p1-3.h39.eulerosv2r8.aarch64
ncurses-devel-6.1-8.20180923.h1.eulerosv2r8.aarch64
readline-devel-7.0-12.eulerosv2r8.aarch64
gdb-gdbserver-8.2-3.h5.eulerosv2r8.aarch64
atlas-devel-3.10.3-6.h8.eulerosv2r8.aarch64
atlas-3.10.3-6.h8.eulerosv2r8.aarch64
atlas-devel-3.10.3-6.h8.eulerosv2r8.aarch64
lapack-devel-3.8.0-10.eulerosv2r8.aarch64
lapack-3.8.0-10.eulerosv2r8.aarch64
lapack-devel-3.8.0-10.eulerosv2r8.aarch64
gcc-gfortran-7.3.0-20190804.h24.eulerosv2r8.aarch64
initscripts-10.01-1.h17.eulerosv2r8.aarch64
freetype-devel-2.9.1-2.eulerosv2r8.aarch64
libpng-devel-1.6.34-6.h3.eulerosv2r8.aarch64
xdg-utils-1.1.3-3.eulerosv2r8.noarch
```

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）：

```
sudo yum install -y gcc gcc-c++ make cmake automake kernel-devel zlib-devel bzip2-devel sqlite-devel openssl-devel libxslt-devel libffi-devel gnome-keyring glib2 dbus-glib-devel xterm openssh-clients openssh-server ncurses-devel readline-devel gdb-gdbserver atlas atlas-devel lapack lapack-devel gcc-gfortran initscripts freetype-devel libpng-devel xdg-utils
```

2. 检查系统是否安装python开发环境。

   ATC安装依赖python3环境，分别使用命令**python3.7.5 --version**、**pip3.7.5 --version**检查是否已经安装，如果分别返回如下信息则说明已经安装。

```
Python 3.7.5
pip 20.3.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

   否则请根据如下方式安装python3.7.5。

   a. 使用wget下载python3.7.5源码包，可以下载到MindStudio安装服务器任意目录，命令为：

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

   b. 进入下载后的目录，解压源码包，命令为：

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

   c. 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
sudo mkdir /usr/local/python3.7.5
./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
make && 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。

d. 执行如下命令复制libpython3.7m.so.1.0动态库：

   i. 由于yum工具强依赖于系统自带的libpython3.7m.so.1.0文件，因此需要备份系统自带的libpython3.7m.so.1.0文件。请执行如下命令：
```
cd /usr/lib64
cp libpython3.7m.so.1.0 libpython3.7m.so.1.0.bak
```
   如果环境上没有/usr/lib64，则以实际路径为准。

   ii. 将编译后的文件libpython3.7m.so.1.0复制到/usr/lib64。
```
sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib64
```
   如果出现如下显示，则输入y，表示覆盖系统自带的libpython3.7m.so.1.0文件。
```
cp: overwrite 'libpython3.7m.so.1.0'?
```
   如果环境上没有/usr/lib64，则复制到/usr/lib目录：
```
sudo cp /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 /usr/lib
```
   libpython3.7m.so.1.0文件所在路径请根据实际情况进行替换。

   iii. 当需要使用yum工具时，需要执行如下命令将备份的文件libpython3.7m.so.1.0.bak恢复。
```
mv libpython3.7m.so.1.0.bak libpython3.7m.so.1.0
```

e. 执行如下命令设置软链接：
```
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/bin/python3.7
sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/bin/pip3.7
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/bin/python3.7.5
sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/bin/pip3.7.5
```
   执行上述软链接时如果提示链接已经存在，则可以先执行如下命令删除原有链接然后重新执行。
```
sudo rm -rf /usr/bin/python3.7
sudo rm -rf /usr/bin/pip3.7
sudo rm -rf /usr/bin/python3.7.5
sudo rm -rf /usr/bin/pip3.7.5
```

f. 设置python3.7.5环境变量。

   i. 以安装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如下内容。
```
#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
```

   ii. 执行**:wq!**命令保存文件并退出。

   iii. 执行**source ~/.bashrc**命令使其立即生效。

g. 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。
```
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7  --version
```

步骤3 使用root用户编译安装matplotlib，普通安装用户直接使用即可。

1. 使用wget下载matplotlib-3.3.3源码包，可以下载到MindStudio安装服务器任意目录，命令为：

```
wget https://files.pythonhosted.org/packages/7b/
b3/7c48f648bf83f39d4385e0169d1b68218b838e185047f7f613b1cfc57947/matplotlib-3.3.3.tar.gz
```

2. 进入下载后的目录，解压源码包，命令为：
```
tar -zxvf matplotlib-3.3.3.tar.gz
```
3. 进入解压后的文件夹，执行安装命令：
```
cd matplotlib-3.3.3
python3.7.5 setup.py build
python3.7.5 setup.py install
```

说明

执行**python3.7.5 setup.py build**命令失败时，请执行以下步骤：

1. 检查目录下是否存在build目录，若无，请执行**mkdir build**命令创建build目录。
2. 执行**cd build**命令进入build目录。
3. 执行**wget –no-check-certificate https://downloads.sourceforge.net/project/freetype/freetype2/2.6.1/freetype-2.6.1.tar.gz**命令下载安装包。
4. 执行**tar -zxvf freetype-2.6.1.tar.gz**命令解压安装包。
5. 重新执行**步骤3.3**。

步骤4 安装 Python3依赖。

安装TBE依赖软件：numpy，decorator；TIK依赖软件：sympy，cffi，gnureadline；utst工具依赖软件：coverage，精度比对依赖protobuf、scipy；profiling工具依赖：protobuf、grpcio、grpcio-tools、requests。若用户使用算子ST用例从模型获取shape等功能，则需要安装tensorflow，否则不用安装。

安装前请先使用**pip3.7.5 list**命令检查是否安装相关依赖，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）：

```
pip3.7.5 install attrs --user
pip3.7.5 install psutil --user
pip3.7.5 install decorator --user
pip3.7.5 install numpy --user
pip3.7.5 install protobuf==3.11.3 --user
pip3.7.5 install scipy --user
pip3.7.5 install sympy --user
pip3.7.5 install cffi --user
pip3.7.5 install gnureadline --user
pip3.7.5 install coverage --user
pip3.7.5 install pylint --user
pip3.7.5 install grpcio --user
pip3.7.5 install grpcio-tools --user
pip3.7.5 install requests --user
pip3.7.5 install xlrd==1.2.0 --user
pip3.7.5 install absl-py --user
pip3.7.5 install pandas --user
pip3.7.5 install pytest --user
```

如果执行上述命令时无法连接网络，且提示“Could not find a version that satisfies the requirement xxx”，请参见**安装-FAQ**解决。

步骤5 安装JDK。

1. 安装JDK之前，先用如下命令检查系统是否已经存在JAVA：
```
which java
```
若返回相关路径，则说明已经存在，否则执行如下命令安装JDK。
```
sudo yum install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel
```
2. 配置JAVA_HOME环境变量，MindStudio的安装及运行都依赖该环境变量，设置方法如下：
   a. 在任何目录下执行如下命令，打开**.bashrc**文件。
```
vi ~/.bashrc
```

b. 在文件的最后一行后面添加如下内容。
```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export PATH=$JAVA_HOME/bin:$PATH
```

说明

“JAVA_HOME”为JDK的安装目录，若用户已经配置了JDK，请根据安装目录的实际情况进行修改。若根据上述步骤安装的JDK，则安装目录不用修改。

c. 执行如下命令保存文件并退出。
```
:wq!
```

d. 执行如下命令使环境变量生效。
```
source ~/.bashrc
```

e. 执行如下命令检查环境变量设置：
```
echo $JAVA_HOME
```
回显信息如下：
```
/usr/lib/jvm/java-1.8.0-openjdk
```

f. 执行如下命令检查JDK安装。
```
which jconsole
```
如果输出如下回显信息表示安装成功，否则表示JDK安装失败。
```
/usr/lib/jvm/java-8-openjdk-arm64/bin/jconsole
```

步骤6 （可选）安装Tensorflow 1.15.0。具体操作请参见**安装Tensorflow 1.15.0**。

步骤7 返回**2.3.4 安装MindStudio**。

----结束

安装 Tensorflow 1.15.0

需要安装Tensorflow 1.15.0才可以进行算子开发验证、训练业务开发。

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。

1. 编译安装HDF5。

   a. 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
   ```

   b. 进入下载后的目录，解压源码包，命令为：
   ```
   tar -zxvf hdf5-1.10.5.tar.gz
   ```

   c. 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
   ```

2. 配置环境变量并建立动态链接库软连接。

   a. 配置环境变量。
   ```
   export CPATH="/usr/include/hdf5/include/:/usr/include/hdf5/lib/"
   ```

   b. 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
   ```

3. 安装h5py。

a. root用户下安装h5py依赖包命令如下。
```
pip3.7 install Cython
```
b. 安装h5py命令如下。
```
pip3.7 install h5py==2.8.0
```

4. 安装grpcio。

   root用户下安装grpcio命令如下。
```
pip3.7 install grpcio==1.32.0
```

安装步骤

1. 下载已经编译好的Tensorflow1.15软件包：**tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl**
2. 安装编译好的Tensorflow。

   进入软件包所在目录，root用户执行如下命令安装Tensorflow1.15：
```
pip3.7 install tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl
```
   非root用户执行如下命令：
```
pip3.7 install tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl --user
```

15.3 算子规格说明

15.3.1 Caffe 框架算子规格

该算子规格仅适用于Caffe框架原生IR定义的网络模型。如果要查看基于Ascend IR定义的单算子信息，请参见《 **CANN算子清单CANN算子清单CANN算子清单CANN算子清单** 》手册。

总体限制说明：

1. 如果无特殊说明，则默认输入和输出的 w<=4096，h<=4096。

支持的Caffe算子请参考**Caffe算子规格清单**。

15.3.2 TensorFlow 框架算子规格

该算子规格仅适用于TensorFlow框架原生IR定义的网络模型。如果要查看基于Ascend IR定义的单算子信息，请参见《 **CANN算子清单CANN算子清单CANN算子清单CANN算子清单** 》手册。

支持的TensorFlow算子请参考**TensorFlow算子规格清单**。

15.3.3 ONNX 算子规格

该算子规格仅适用于ONNX原生IR定义的网络模型。如果要查看基于Ascend IR定义的单算子信息，请参见《 **CANN算子清单CANN算子清单CANN算子清单CANN算子清单** 》手册。

支持的ONNX算子请参考**ONNX算子规格清单**。

15.4 特性配置

15.4.1 定制网络修改（Caffe）

15.4.1.1 简介

本章节修改只适用于Caffe网络模型。

网络的算子可以分为如下几类：

- 标准算子：昇腾AI处理器支持的Caffe标准算子，比如Convolution等。
- 扩展算子：昇腾AI处理器支持的公开但非Caffe标准算子，分为 2 种：
  - 一种是基于Caffe框架进行自定义扩展的算子，比如Faster RCNN中的ROIPooling、SSD中的归一化算子Normalize等。
  - 另外一种是来源于其他深度学习框架的自定义算子，比如YOLOv2中Passthrough等。

Faster RCNN、SSD等网络模型都包含了一些原始Caffe框架中没有定义的算子结构，如ROIPooling、Normalize、PSROI Pooling和Upsample等。为了使昇腾AI处理器能支持这些网络，需要对原始的Caffe框架网络模型进行扩展，降低开发者开发自定义算子/开发后处理代码的工作量。若开发者的Caffe框架网络模型中使用了这些扩展算子，在进行模型转换时，需要先在prototxt中修改/添加扩展层的定义，才能成功进行模型转换。

本章节提供了昇腾AI处理器的扩展算子列表，并给出了如何根据扩展算子修改prototxt文件方法。

15.4.1.2 扩展算子列表

表 15-2 扩展算子列表

分类	算子类型	说明
过程算子	**Reverse**	将输入tensor指定的轴进行反转。
	ROIPooling	用于Faster R-CNN中，对Roi（Region of interest）进行池化操作，主要用于目标检测任务。
	PSROIPooling	用于R-FCN中，进行位置敏感的候选区域池化操作，主要用于目标检测任务。
	Upsample	使用pooling mask的上采样。 用于yolo网络。
	Normallize	将SSD网络中Channel维度中的元素在L2进行归一化。

分类	算子类型	说明
	Reorg	Reorg layer in Darknet，将通道数据转移到平面上，或反过来操作。 **对应算子规格中的PassThrough算子。**
	Proposal	用于Faster R-CNN，根据rpn_cls_prob的foreground，rpn_bbox_pred中的bounding box regression修正anchors获得精确的proposals。
	ROIAlign	从feature map中获取ROI（range of interest）的特征矩阵。
	ShuffleChannel	把channel维度分为[group, channel/Group]，再在channel维度中进行数据转置[channel/Group,group]。
	Yolo（Yolo/Detection/Region）	Yolo/Detection/Region算子，都需要替换为Yolo算子，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	PriorBox	用于SSD网络，根据输入参数，生成prior box。
	SpatialTransformer	用于仿射变换。
后处理算子	YoloV3DetectionOutput	此算子用于YOLOv3的后处理过程，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	YoloV2DetectionOutput	此算子用于YOLOv2的后处理过程，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	SSDDetectionOutput	此算子用于SSD网络的后处理过程，用于整合预选框、预选框偏移以及得分三项结果，最终输出满足条件的目标检测框、目标的label和得分。

分类	算子类型	说明
	FSRDetectionOutput	此算子用于Faster R-CNN网络的后处理过程，对结果进行分类，并对每个类输出最终的bbox数量、坐标、类别概率及类别索引。

15.4.1.3 扩展算子规则

扩展算子的实现可以在Caffe、TensorFlow、MindSpore深度学习框架中实现，扩展算子中的第一种是基于Caffe框架进行扩展实现的，如ROIPooling、Normalize、PSROIPolling和Upsample层等，因此有公开的prototxt标准定义。而扩展算子中的第二种并不是在Caffe框架下实现的，对于这类网络中的自定义算子，也需要给出prototxt的标准定义，用于在prototxt中定义网络中相应的算子。

Reverse

在指定的轴上反序，如输入为[1,2,3]，其反序为[3,2,1]。

算子定义如下：

1. 在LayerParameter中添加ReverseParameter
```
message LayerParameter {
...
optional ReverseParameter reverse_param = 157;
...
}
```
2. 定义ReverseParameter类型以及属性参数
```
message ReverseParameter{
  repeated int32 axis = 1;
}
```

ROIPooling

在目标检测算法中，region proposal产生的ROI大小不一，而分类网络的FC层要固定的输入，所以ROIPooing起到一个连接作用。

ROIPooling层为Faster RCNN网络中用于将不同图像经过卷积层后得到的feature map进行维度统一的操作，输入为一个feature map以及需要从中提取的ROI框坐标值，输出为一个维度归一化的feature map。

ROIPooling层需要对caffe.proto文件进行扩展，定义ROIPoolingParameter，扩展方式如下所示，定义的参数包括：

- spatial_scale，即输入feature map与原始图片的尺寸比例，用于转换ROI的坐标，因为ROI坐标是在原图尺寸上的。
- pooled_h、pooled_w，输出feature map在spatial维度上h和w的大小。

1. 在LayerParameter中添加ROIPoolingParameter
```
message LayerParameter {
...
optional ROIPoolingParameter roi_pooling_param = 161;
...
}
```

2. 定义ROIPoolingParameter类型以及属性参数

```
message ROIPoolingParameter {
    required int32 pooled_h = 1;
    required int32 pooled_w = 2;
    optional float spatial_scale = 3 [default=0.0625];
    optional float spatial_scale_h = 4;
    optional float spatial_scale_w = 5;
}
```

基于上述原则，ROIPooling层在prototxt中定义的代码样例如下：

```
layer {
name: "roi_pooling"
type: "ROIPooling"
bottom: "res4f"
bottom: "rois"
bottom: "actual_rois_num"
top: "roi_pool"
roi_pooling_param {
  pooled_h: 14
  pooled_w: 14
  spatial_scale:0.0625
  spatial_scale_h:0.0625
  spatial_scale_w:0.0625
  }
}
```

PSROIPooling

PSROIPooling层的操作与ROIPooling层类似，不同之处在于不同空间维度输出的图片特征来自不同的feature map channels，且对每个小区域进行的是Average Pooling，不同于ROIPooling的Max Pooling。

对于一个输出 k*k 的结果，不同空间维度的特征取自输入feature map中不同的组，即将输入的feature map均匀分为k*k组，每组的channel数与输出的channel一致，得到上述输出。

PSROIPooling层需要对caffe.proto文件进行扩展，定义PSROIPoolingParameter，扩展方式如下所示，定义的参数包括：

- spatial_scale，即输入feature map与原始图片的尺寸比例。
- output_dim，输出feature map的channel。
- group_size，输出的spatial维度，即上述的k。

1. 在LayerParameter中添加PSROIPoolingParameter

```
message LayerParameter {
...
optional PSROIPoolingParameter psroi_pooling_param = 207;
...
}
```

2. 定义PSROIPoolingParameter类型以及属性参数

```
message PSROIPoolingParameter {
   required float spatial_scale = 1;
   required int32 output_dim = 2; // output channel number
   required int32 group_size = 3; // number of groups to encode position-sensitive score maps
}
```

基于上述原则，PSROIPooling层在prototxt中定义的代码样例如下：

```
layer {
  name: "psroipooling"
  type: "PSROIPooling"
  bottom: "some_input"
```

```
  bottom: "some_input"
  top: "some_output"
  psroi_pooling_param {
    spatial_scale: 0.0625
    output_dim: 21
    group_size: 7
  }
}
```

Upsample

Upsample层为Pooling层的逆操作，其中每个Upsample层均与网络之前一个对应大小输入、输出Pooling层一一对应，完成feature map在spatial维度上的扩充。

Upsample层需要对caffe.proto文件进行扩展，定义UpsampleParameter，扩展方式示例如下所示。 定义的参数包括stride，即输出与输入的尺寸比例，如2。

1. 在LayerParameter中添加UpsampleParameter

```
message LayerParameter {
...
optional UpsampleParameter upsample_param = 160;
...
}
```

2. 定义UpsampleParameter类型以及属性参数

```
message UpsampleParameter{
    optional float scale = 1[default = 1];
    optional int32 stride = 2[default = 2];
    optional int32 stride_h = 3[default = 2];
    optional int32 stride_w = 4[default=2];
}
```

基于上述原则，Upsample在prototxt中定义的代码样例如下：

```
layer {
    name: "layer86-upsample"
    type: "Upsample"
    bottom: "some_input"
    top: "some_output"
    upsample_param {
        scale: 1
                stride: 2
    }
}
```

Normallize

Normalize层为SSD网络中的一个归一化层，主要作用是将空间或者通道内的元素归一化到0到1之间，其进行的操作为对于一个c*h*w的三维tensor，输出是同样大小的tensor，其中间计算为每个元素以channel方向的平方和的平方根求 normalize，其具体计算公式为：

$$x_i' = \frac{x_i}{\sqrt{\sum_{i=1}^{c} x_i^2}}$$

其中分母位置的平方和的累加向量为同一h与w位置的所有c方向的向量，如**图15-1**中的橙色区域。

图 15-1 Normalize 层原理图

经过上述计算归一化后，再对每个feature map做scale，每个channel对应一个scale值。

Normalize层需要对caffe.proto文件进行扩展，定义NormalizeParameter，扩展方式如下所示。定义的参数包括：

- across_spatial参数表示是否对整个图片进行归一化，归一化的维度为：1 x c x h x w，否则对每个像素点进行归一化：1 x c x 1 x 1。
- channels_shared表示scale是否相同，如果为true，则scale都是一样的，否则对于channel一样，对不同channel像素点是不一样的，默认为True。
- eps防止normalize过程中除以0的一个很小数值，默认为1e-10，可配置。

normalize的计算公式转换为：

$$x_i = \frac{x_i}{\left(\sum_1^n x_i^2 + \text{eps}\right)^{\frac{1}{2}}} \times \text{scale}_i$$

算子定义如下：

1. 在LayerParameter中添加NormalizeParameter
```
message LayerParameter {
...
optional NormalizeParameter norm_param = 206;
...
}
```
2. 定义NormalizeParameter类型以及属性参数
```
message NormalizeParameter {
  optional bool across_spatial = 1 [default = true];
  // Initial value of scale. Default is 1.0 for all
  optional FillerParameter scale_filler = 2;
  // Whether or not scale parameters are shared across channels.
  optional bool channel_shared = 3 [default = true];
  // Epsilon for not dividing by zero while normalizing variance
  optional float eps = 4 [default = 1e-10];
}
```

基于上述原则，Normalize在prototxt中定义的代码样例如下：

```
layer {
  name: "normalize_layer"
  type: "Normalize"
  bottom: ""some_input"
```

```
  top: "some_output"
  norm_param {
    across_spatial: false
    scale_filler {
      type: "constant"
      value: 20
    }
    channel_shared: false
  }
}
```

Reorg

Reorg算子在昇腾AI处理器内部以PassThrough算子呈现，将通道数据转移到平面上，或反过来操作。

PassThrough层为Yolo v2中的一个自定义层，由于Yolo v2并不是使用Caffe框架实现，因此对于该层没有标准的定义。该层实现的功能为将feature map在spatial维度上的数据展开到channel维度上，原始在channel维度上连续的元素在展开后的feature map中依然是连续的。

算子定义如下：

1. 在LayerParameter中添加ReorgParameter

```
message LayerParameter {
...
 optional ReorgParameter reorg_param = 155;
...
}
```

2. 定义ReorgParameter类型以及属性参数

```
message ReorgParameter{
    optional uint32 stride = 2 [default = 2];
    optional bool reverse = 1 [default = false];
}
```

基于上述原则，Reorg在prototxt中定义的代码样例如下：

```
layer {
    bottom: "some_input"
    top: "some_output"
    name: "reorg"
    type: "Reorg"
    reorg_param {
        stride: 2
    }
}
```

Proposal

proposal算子根据rpn_cls_prob的foreground，rpn_bbox_pred中的bounding box regression修正anchors获得精确的proposals。

具体可以分为3个算子decoded_bbox、topk和nms，实现如图15-2所示。

图 15-2 proposal 算子实现

算子定义如下：

1. 在LayerParameter中添加ProposalParameter

```
message LayerParameter {
...
 optional ProposalParameter proposal_param = 201;
...
}
```

2. 定义ProposalParameter类型以及属性参数

```
message ProposalParameter {
    optional float feat_stride = 1 [default = 16];
    optional float base_size = 2 [default = 16];
    optional float min_size = 3 [default = 16];
    repeated float ratio = 4;
    repeated float scale = 5;
    optional int32 pre_nms_topn = 6 [default = 3000];
    optional int32 post_nms_topn = 7 [default = 304];
    optional float iou_threshold = 8 [default = 0.7];
    optional bool output_actual_rois_num = 9 [default = false];
}
```

基于上述原则，Proposal在prototxt中定义的代码样例如下：

```
layer {
name: "faster_rcnn_proposal"
type: "Proposal"                    // 算子Type


bottom: "rpn_cls_prob_reshape"
bottom: "rpn_bbox_pred"
bottom: "im_info"
top: "rois"
top: "actual_rois_num"        // 增加的算子输出
  proposal_param {

  feat_stride: 16
  base_size: 16
  min_size: 16
  pre_nms_topn: 3000
  post_nms_topn: 304
  iou_threshold: 0.7
  output_actual_rois_num: true
  }
}
```

ROIAlign

ROI Align 是在Mask-RCNN论文里提出的一种区域特征聚集方式, 与ROIPooling算法进行改进：用双线性插值替换ROI Pooling操作中两次量化，以解决ROIPooling造成的区域不匹配的问题，提高检测准确性。

ROIAlign算子是从feature map中获取ROI（range of interest），分为pooled_h x pooled_h 个单元格，每个单元格均分为sampling_ratio*sampling_ratio个小方格，每个小方格的中心点就是采样点。如图15-3所示，虚线部分表示feature map，实线表示ROI，这里将ROI切分成2x2的单元格。如果采样点数是4，则首先将每个单元格子均分成四个小方格（如红色线所示），每个小方格中心就是采样点。由于采样点的坐标通常是浮点数，所以需要对采样点像素进行双线性插值（如图15-3中的四个箭头所示），就可以得到该像素点的值了。然后对每个单元格内的四个采样点取均值，就可以得到最终的ROIAlign的结果。

图 15-3 ROIAlign 原理图

算子定义如下：

1. 在LayerParameter中添加ROIAlignParameter
```
message LayerParameter {
...
 optional ROIAlignParameter  roi_align_param = 154;
...
}
```
2. 定义ROIAlignParameter类型以及属性参数
```
message ROIAlignParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
  optional int32 sampling_ratio = 4 [default = -1];
  optional int32 roi_end_mode = 5 [default = 0];
}
```

根据上述类型以及属性用户可以自定义prototxt。

ShuffleChannel

ShuffleChannel是把channel维度分为[group, channel/Group]，然后再在channel维度中进行数据转置[channel/Group,group]。

例如对于channel=4，group=2，则执行此算子后，就是把channel[1]、channel[2]的数据交换位置。

算子定义如下：

1. 在LayerParameter中添加ShuffleChannelParameter
```
message LayerParameter {
...
 optional ShuffleChannelParameter shuffle_channel_param = 159;
...
}
```
2. 定义ShuffleChannelParameter类型以及属性参数
```
message ShuffleChannelParameter{
    optional uint32 group = 1[default = 1]; // The number of group
}
```

基于上述原则，ShuffleChannel在prototxt中定义的代码样例如下：

```
layer {
  name: "layer_shuffle"
  type: "ShuffleChannel"
  bottom: "some_input"
  top: "some_output"
  shuffle_channel_param {
    group: 3
  }
}
```

Yolo

YOLO算子出现在YOLO V2网络，且目前仅在YOLO V2、V3网络中使用，对数据做sigmoid和softmax操作。

- 在YOLO V2中，根据backgroud和softmax的参数，有4种场景：
  a. background=false，softmax=true，
     对(x,y,h,w)中的(x,y)做sigmoid，对b做sigmoid，对classes做softmax。
  b. background=false，softmax=false，
     对(x,y,h,w)中的(x,y)做sigmoid，对b做sigmoid，对classes做sigmoid。
  c. background=true，softmax=false，
     对(x,y,h,w)中的(x,y)做sigmoid，对b不做计算，对classes做sigmoid。
  d. background=true，softmax= true，
     对(x,y,h,w)中的(x,y)做sigmoid，对b和classes放在一起做softmax。
- 在YOLO V3中，只有一种场景：对(x,y,h,w)中的(x,y)做sigmoid，对b做sigmoid，对classes做sigmoid。

输入数据格式为Tensor(n, coords+backgroup+classes,l.h,l.w)，其中n是anchor box的数量，corrds表示x,y,w,h。

算子定义如下：

1. 在LayerParameter中添加YoloParameter
```
message LayerParameter {
...
 optional YoloParameter yolo_param = 199;
...
}
```
2. 定义YoloParameter类型以及属性参数
```
message YoloParameter {
  optional int32 boxes = 1 [default = 3];
  optional int32 coords = 2 [default = 4];
  optional int32 classes = 3 [default = 80];
  optional string yolo_version = 4 [default = "V3"];
  optional bool softmax = 5 [default = false];
```

```
    optional bool background = 6 [default = false];
    optional bool softmaxtree = 7 [default = false];
  }
```

基于上述原则，Yolo在prototxt中定义的代码样例如下：

```
layer {
    bottom: "layer82-conv"
    top: "yolo1_coords"
    top: "yolo1_obj"
    top: "yolo1_classes"
    name: "yolo1"
    type: "Yolo"
    yolo_param {
        boxes: 3
        coords: 4
        classes: 80
        yolo_version: "V3"
        softmax: true
        background: false
    }
}
```

PriorBox

根据输入的参数，生成prior box。

下面以conv7_2_mbox_priorbox为例，根据对应的参数生成prior box的数量。定义如下：

```
layer{
    name:"conv7_2_mbox_priorbox"
    type:"PriorBox"
    bottom:"conv7_2"
    bottom:"data"
    top:"conv7_2_mbox_priorbox"
    prior_box_param{
        min_size:162.0
        max_size:213.0
        aspect_ratio:2
        aspect_ratio:3
        flip:true
        clip:false
        variance:0.1
        variance:0.1
        variance:0.2
        variance:0.2
        img_size:300
        step:64
        offset:0.5
    }
}
```

1. 宽高都为minsize生成prior box。
2. 如果存在max_size, 则用sqrt(min_size*max_size)确定宽高生成一个框(约束，max_size>min_size)。
3. 根据aspect_ratio(如定义所示aspect_ratio为2，3，flip是true，自动添加aspect_ratio=1/2、1/3)，生成对应的prior box。

因此，num_priors_(prior box的数量)= min_size的数量+aspect_ratio的数量（这里为4）*min_size的数量（这里为1）+max_size的数量 （max_size的数量和min_size 的数量一一对应）。

算子定义如下：

1. 在LayerParameter中添加PriorBoxParameter

```
message LayerParameter {
...
 optional PriorBoxParameter prior_box_param = 203;
...
}
```

2. 定义PriorBoxParameter类型以及属性参数

```
message PriorBoxParameter {
  // Encode/decode type.
  enum CodeType {
    CORNER = 1;
    CENTER_SIZE = 2;
    CORNER_SIZE = 3;
  }
  // Minimum box size (in pixels). Required!
  repeated float min_size = 1;
  // Maximum box size (in pixels). Required!
  repeated float max_size = 2;
  // Various of aspect ratios. Duplicate ratios will be ignored.
  // If none is provided, we use default ratio 1.
  repeated float aspect_ratio = 3;
  // If true, will flip each aspect ratio.
  // For example, if there is aspect ratio "r",
  // we will generate aspect ratio "1.0/r" as well.
  optional bool flip = 4 [default = true];
  // If true, will clip the prior so that it is within [0, 1]
  optional bool clip = 5 [default = false];
  // Variance for adjusting the prior bboxes.
  repeated float variance = 6;
  // By default, we calculate img_height, img_width, step_x, step_y based on
  // bottom[0] (feat) and bottom[1] (img). Unless these values are explicitely
  // provided.
  // Explicitly provide the img_size.
  optional uint32 img_size = 7;
  // Either img_size or img_h/img_w should be specified; not both.
  optional uint32 img_h = 8;
  optional uint32 img_w = 9;

  // Explicitly provide the step size.
  optional float step = 10;
  // Either step or step_h/step_w should be specified; not both.
  optional float step_h = 11;
  optional float step_w = 12;

  // Offset to the top left corner of each cell.
  optional float offset = 13 [default = 0.5];
}
```

基于上述原则，PriorBox在prototxt中定义的代码样例如下：

```
layer {
  name: "layer_priorbox"
  type: "PriorBox"
  bottom: "some_input"
  bottom: "some_input"
  top: "some_output"
  prior_box_param {
    min_size: 30.0
    max_size: 60.0
    aspect_ratio: 2
    flip: true
    clip: false
    variance: 0.1
    variance: 0.1
    variance: 0.2
    variance: 0.2
    step: 8
    offset: 0.5
```

```
 }
}
```

SpatialTransformer

该算子计算过程其实做了一个仿射变换：仿射变换的参数，可以是在prototxt固定的，多个batch使用一份；也可以是层的第二个输入，每个batch使用不一样的参数。

计算步骤：

1. 对于输出坐标，通过如下公式将其变换成[-1,1]区间中的值。

$$x' = x \times \frac{1.0}{\mathrm{out}_h} \times 2 - 1$$

$$y' = y \times \frac{1.0}{\mathrm{out}_w} \times 2 - 1$$

计算代码如下：

```
Dtype* data = output_grid.mutable_cpu_data();
        for(int i=0; i< output_H_ * output_W_; ++i) {
                data[3 * i] = (i / output_W_) * 1.0 / output_H_ * 2 - 1;
                data[3 * i + 1] = (i % output_W_) * 1.0 / output_W_ * 2 - 1;
                data[3 * i + 2] = 1;
        }
```

2. 通过仿射变换，转换为输入的坐标，其中s为输入坐标，t为输出坐标，计算公式如下：

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

计算代码如下：

```
caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasTrans, output_H_ * output_W_, 2, 3, (Dtype)1.,
    output_grid_data, full_theta_data + 6 * i, (Dtype)0., coordinates);
```

3. 通过输入坐标取对应位置的值，赋值给输出的对应位置。

因为在第一步对输出坐标做了一次转换，所以对应的输入坐标，使用同样的方式再缩放回去，代码样例如下：

```
Dtype x = (px + 1) / 2 * H;
Dtype y = (py + 1) / 2 * W;
        if(debug) std::cout<<prefix<<"(x, y) = ("<<x<<", "<<y<<")"<<std::endl;

  for(int m = floor(x); m <= ceil(x); ++m)
    for(int n = floor(y); n <= ceil(y); ++n) {
            if(debug) std::cout<<prefix<<"(m, n) = ("<<m<<", "<<n<<")"<<std::endl;
            if(m >= 0 && m < H && n >= 0 && n < W) {
                    res += (1 - abs(x - m)) * (1 - abs(y - n) * pic[m * W + n]);
                    if(debug) std::cout<<prefix<<" pic[m * W + n]= "<<std::endl;
            }
    }
```

算子定义如下：

1. 在LayerParameter中添加SpatialTransformParameter

```
message LayerParameter {
...
 optional SpatialTransformParameter spatial_transform_param = 153;
...
}
```

2. 定义SpatialTransformParameter类型以及属性参数

```
message SpatialTransformParameter {
  optional uint32 output_h = 1 [default = 0];
  optional uint32 output_w = 2 [default = 0];
  optional float border_value = 3 [default = 0];
  repeated float affine_transform = 4;
  enum Engine {
    DEFAULT = 0;
    CAFFE = 1;
    CUDNN = 2;
  }
  optional Engine engine = 15 [default = DEFAULT];
}
```

基于上述原则，SpatialTransform层在prototxt中定义的代码样例如下：

```
layer {
  name: "st_1"
  type: "SpatialTransformer"
  bottom: "data"
  bottom: "theta"
  top: "transformed"
  st_param {
    to_compute_dU: false
    theta_1_1: -0.129
    theta_1_2: 0.626
    theta_2_1: 0.344
    theta_2_2: 0.157
  }
}
```

15.4.1.4 样例参考

本章节介绍几种常用网络的修改方法。

15.4.1.4.1 FasterRCNN 网络模型 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

如下以FasterRCNN Resnet34网络模型为例进行说明。

1. proposal算子修改。

   根据**15.3.1 Caffe框架算子规格**，该算子有3个输入，2个输出；根据上述原则，对原始proposal算子进行修改，type修改为caffe.proto文件中的类型，增加actual_rois_num输出节点。参见caffe.proto文件中的属性定义增加相应属性信息。修改情况如**图15-4**所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 15-4 原始算子与修改后的算子比对 1

代码样例如下：

```
layer {
name: "faster_rcnn_proposal"
type: "Proposal"                    // 算子Type


bottom: "rpn_cls_prob_reshape"
bottom: "rpn_bbox_pred"
bottom: "im_info"
top: "rois"
top: "actual_rois_num"        // 增加的算子输出
  proposal_param {

  feat_stride: 16
  base_size: 16
  min_size: 16
  pre_nms_topn: 3000
  post_nms_topn: 304
  iou_threshold: 0.7
  output_actual_rois_num: true
  }
}
```

参数解释请参见**15.3.1 Caffe框架算子规格**。

2. 最后一层增加FSRDetectionOutput算子，用于输出最终的检测结果。

   对于FasterRCNN网络，参考**15.4.1.2 扩展算子列表**在原始prototxt文件的最后增加后处理算子层FSRDetectionOutput ，参见**15.3.1 Caffe框架算子规格**，FSRDetectionOutput算子有五个输入，两个输出，此算子的Type及属性定义如下：

   代码样例如下：

```
layer {
  name: "FSRDetectionOutput_1"
  type: "FSRDetectionOutput"

  bottom: "rois"
  bottom: "bbox_pred"
  bottom: "cls_prob"
  bottom: "im_info"
  bottom: "actual_rois_num"
  top: "actual_bbox_num1"
  top: "box1"

  fsrdetectionoutput_param {

    num_classes:3
    score_threshold:0.0
    iou_threshold:0.7
    batch_rois:1
  }
}
```

参数解释请参见15.3.1 Caffe框架算子规格。

15.4.1.4.2 YOLOv3 网络模型 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

1. upsample算子upsample_param属性参数修改。

   参见15.3.1 Caffe框架算子规格，需要将原始算子prototxt中的scale:2修改为scale:1 stride:2。

   对比情况如图15-5所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

   图 15-5 原始算子与修改后的算子比对 4

   参数解释请参见15.3.1 Caffe框架算子规格。

2. 新增三个Yolo算子。

   由于Yolo算子+DetectionOutput算子才是特征检测网络完整的后处理逻辑，根据原始算子prototxt所示，在增加YoloV3DetectionOutput算子之前需要先增加三个Yolo算子。

   根据15.3.1 Caffe框架算子规格，该算子有1个输入，3个输出，根据上述原则，构造的Yolo算子代码样例如下：

   – 算子1代码样例：

   ```
   layer {
      bottom: "layer82-conv"
      top: "yolo1_coords"
      top: "yolo1_obj"
      top: "yolo1_classes"
      name: "yolo1"
      type: "Yolo"
      yolo_param {
         boxes: 3
         coords: 4
         classes: 80
         yolo_version: "V3"
         softmax: true
         background: false
      }
   }
   ```

   – 算子2代码样例：

   ```
   layer {
      bottom: "layer94-conv"
      top: "yolo2_coords"
      top: "yolo2_obj"
      top: "yolo2_classes"
      name: "yolo2"
      type: "Yolo"
      yolo_param {
         boxes: 3
         coords: 4
         classes: 80
   ```

```
        yolo_version: "V3"
        softmax: true
        background: false
    }
}
```

– 算子3代码样例：

```
layer {
    bottom: "layer106-conv"
    top: "yolo3_coords"
    top: "yolo3_obj"
    top: "yolo3_classes"
    name: "yolo3"
    type: "Yolo"
    yolo_param {
        boxes: 3
        coords: 4
        classes: 80
        yolo_version: "V3"
        softmax: true
        background: false
    }
}
```

参数解释请参见**15.3.1 Caffe框架算子规格**。

3. 最后一层增加YoloV3DetectionOutput算子。

   对于YOLOv3网络，参考**15.4.1.2 扩展算子列表**在原始prototxt文件的最后增加后处理算子层YoloV3DetectionOutput ，参见**15.3.1 Caffe框架算子规格**，YoloV3DetectionOutput算子有十个输入，两个输出；根据该原则，构造的后处理算子代码样例如下：

```
layer {
    name: "detection_out3"
    type: "YoloV3DetectionOutput"
    bottom: "yolo1_coords"
    bottom: "yolo2_coords"
    bottom: "yolo3_coords"
    bottom: "yolo1_obj"
    bottom: "yolo2_obj"
    bottom: "yolo3_obj"
    bottom: "yolo1_classes"
    bottom: "yolo2_classes"
    bottom: "yolo3_classes"
    bottom: "img_info"
    top: "box_out"
    top: "box_out_num"
    yolov3_detection_output_param {
            boxes: 3
            classes: 80
            relative: true
            obj_threshold: 0.5
            score_threshold: 0.5
            iou_threshold: 0.45
            pre_nms_topn: 512
            post_nms_topn: 1024
            biases_high: 10
            biases_high: 13
            biases_high: 16
            biases_high: 30
            biases_high: 33
            biases_high: 23
            biases_mid: 30
            biases_mid: 61
            biases_mid: 62
            biases_mid: 45
            biases_mid: 59
            biases_mid: 119
            biases_low: 116
            biases_low: 90
```

```
                    biases_low: 156
                    biases_low: 198
                    biases_low: 373
                    biases_low: 326
    }
}
```

参数解释请参见15.3.1 Caffe框架算子规格。

4. 新增输入：

   由于YoloV3DetectionOutput算子有img_info输入，故模型输入时增加该输入。对比情况如图15-6所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 15-6 原始算子与修改后的算子比对 3

代码样例如下，参数为[batch,4]，4表示netH、netW、scaleH、scaleW四个维度。其中netH,netW为网络模型输入的HW，scaleH,scaleW为原始图片的HW。

```
input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

15.4.1.4.3 YOLOv2 网络模型 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

1. 修改Region算子。

   由于Yolo算子+DetectionOutput算子才是特征检测网络完整的后处理逻辑，在增加YoloV2DetectionOutput算子之前需要将Region过程算子，替换为Yolo算子。

   参见15.3.1 Caffe框架算子规格，Yolo算子有一个输入，三个输出；基于该原则，修改后的Yolo算子如下，对比情况如图15-7所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 15-7 原始算子与修改后的算子比对 2

构造的代码样例如下：

```
layer {
    bottom: "layer31-conv"
    top: "yolo_coords"
    top: "yolo_obj"
    top: "yolo_classes"
    name: "yolo"
    type: "Yolo"
    yolo_param {
        boxes: 5
        coords: 4
        classes: 80
        yolo_version: "V2"
        softmax: true
        background: false
    }
}
```

参数解释请参见**15.3.1 Caffe框架算子规格**。

2. 最后一层增加YoloV2DetectionOutput算子。

   对于YoloV2网络，参考**15.4.1.2 扩展算子列表**在原始prototxt文件的最后增加后处理算子层YoloV2DetectionOutput ，参见**15.3.1 Caffe框架算子规格**，YoloV2DetectionOutput算子有四个输入，两个输出；根据上述原则，构造的后处理算子代码样例如下：

```
layer {
name: "detection_out2"
type: "YoloV2DetectionOutput"
bottom: "yolo_coords"
bottom: "yolo_obj"
bottom: "yolo_classes"
bottom: "img_info"
top: "box_out"
top: "box_out_num"
yolov2_detection_output_param {
  boxes: 5
  classes: 80
  relative: true
  obj_threshold: 0.5
  score_threshold: 0.5
  iou_threshold: 0.45
  pre_nms_topn: 512
  post_nms_topn: 1024
  biases: 0.572730
  biases: 0.677385
  biases: 1.874460
  biases: 2.062530
  biases: 3.338430
  biases: 5.474340
  biases: 7.882820
  biases: 3.527780
  biases: 9.770520
  biases: 9.168280
  }
}
```

   参数解释请参见**15.3.1 Caffe框架算子规格**。

3. 新增输入：

   由于YoloV2DetectionOutput算子有img_info输入，故模型输入时增加该输入。对比情况如**图15-8**所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 15-8 原始算子与修改后的算子比对 1

代码样例如下，参数为[batch,4]，4表示netH、netW、scaleH、scaleW四个维度。其中netH,netW为网络模型输入的HW，scaleH,scaleW为原始图片的HW。

```
input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

15.4.1.4.4 SSD 网络模型 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

对于SSD网络，参考**15.4.1.2 扩展算子列表**在原始prototxt文件的最后增加后处理算子层SSDDetectionOutput。

参见caffe.proto文件（该文件路径为${install_path}/atc/include/proto该文件路径为${install_path}/fwkacllib/include/proto），先在LayerParameter message中添加自定义层参数的声明（如下自定义层在caffe.proto中已经声明，用户无需再次添加）：

```
message LayerParameter {
...
  optional SSDDetectionOutputParameter ssddetectionoutput_param = 232;
...
}
```

参见caffe.proto文件，此算子的Type及属性定义如下：

```
message SSDDetectionOutputParameter {
    optional int32 num_classes= 1 [default = 2];
    optional bool share_location = 2 [default = true];
    optional int32 background_label_id = 3 [default = 0];
    optional float iou_threshold = 4 [default = 0.45];
    optional int32 top_k = 5 [default = 400];
    optional float eta = 6 [default = 1.0];
    optional bool variance_encoded_in_target = 7 [default = false];
    optional int32 code_type = 8 [default = 2];
    optional int32 keep_top_k = 9 [default = 200];
    optional float confidence_threshold = 10 [default = 0.01];
}
```

参见**15.3.1 Caffe框架算子规格**，SSDDetectionOutput算子有3个输入，2个输出，基于上述原则，构造的代码样例如下：

```
layer {
  name: "detection_out"
  type: "SSDDetectionOutput"
  bottom: "bbox_delta"
  bottom: "score"
  bottom: "anchors"
  top: "out_boxnum"
```

```
  top: "y"
  ssddetectionoutput_param {
    num_classes: 2
    share_location: true
    background_label_id: 0
    iou_threshold: 0.45
    top_k: 400
    eta: 1.0
    variance_encoded_in_target: false
    code_type: 2
    keep_top_k: 200
    confidence_threshold: 0.01
  }
}
```

- bottom输入中的bbox_delta对应caffe原始网络中的mbox_loc，score对应caffe原始网络中的mbox_conf_flatten，anchors对应caffe原始网络中的mbox_priorbox；num_classes取值需要与原始网络模型中的取值保持一致。
- top输出多batch场景下:
  - out_boxnum输出shape是(batchnum,8)，每个batchnum的第一个值是实际框的个数。
  - y输出shape是(batchnum,len,8)，其中len是keep_top_k 128对齐后的取值（如batch为2，keep_top_k 为200，则最后输出shape为(2,256,8)），前256*8个数据为第一个batch的结果。

参数解释请参见**15.3.1 Caffe框架算子规格**。

15.4.1.4.5 BatchedMatMul 算子 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

该算子用户输出两个张量的乘积：y=x1*x2（x1和x2的张量相乘，x1和x2维数必须大于2而小于等于8）。如果网络模型使用该算子，则请参见该章节，修改相应prototxt后，再进行模型转换。

参见caffe.proto文件（该文件路径为${install_path}/atc/include/proto该文件路径为${install_path}/fwkacllib/include/proto），先在LayerParameter message中添加自定义层参数的声明（如下自定义层在caffe.proto中已经声明，用户无需再次添加）：

```
message LayerParameter {
...
  optional BatchMatMulParameter batch_matmul_param = 235;
...
}
```

参见caffe.proto文件，此算子的Type及属性定义如下：

```
message BatchMatMulParameter{
    optional bool adj_x1 = 1 [default = false];
    optional bool adj_x2 = 2 [default = false];
}
```

参见**15.3.1 Caffe框架算子规格**，BatchedMatMul算子有2个输入，1个输出，基于上述原则，构造的代码样例如下：

```
layer {
  name: "batchmatmul"
  type: "BatchedMatMul"
  bottom: "matmul_data_1"
```

```
  bottom: "matmul_data_2"
  top: "batchmatmul_1"
  batch_matmul_param {
  adj_x1:false
  adj_x2:true
}
```

参数解释请参见15.3.1 Caffe框架算子规格。

15.4.1.4.6 SENet 网络模型 prototxt 修改

说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

该网络模型中的Axpy算子，需要修改为Reshape、Scale、Eltwise三个算子，修改样例如下：

```
layer {
  name: "conv3_1"
  type: "Axpy"
  bottom: "conv3_1_1x1_up"
  bottom: "conv3_1_1x1_increase"
  bottom: "conv3_1_1x1_proj"
  top: "conv3_1"
}
```

```
layer {
  name: "conv3_1_axpy_reshape"
  type: "Reshape"
  bottom: "conv3_1_1x1_up"
  top: "conv3_1_axpy_reshape"
  reshape_param {
    shape {
      dim: 0
      dim: -1
    }
  }
}
layer {
  name: "conv3_1_axpy_scale"
  type: "Scale"
  bottom: "conv3_1_1x1_increase"
  bottom: "conv3_1_axpy_reshape"
  top: "conv3_1_axpy_scale"
  scale_param {
    axis: 0
    bias_term: false
  }
}
layer {
  name: "conv3_1_axpy_eltwise"
  type: "Eltwise"
  bottom: "conv3_1_axpy_scale"
  bottom: "conv3_1_1x1_proj"
  top: "conv3_1"
}
```

修改后的代码样例如下：

```
layer {
  name: "conv3_1_axpy_reshape"
  type: "Reshape"
  bottom: "conv3_1_1x1_up"
  top: "conv3_1_axpy_reshape"
  reshape_param {
    shape {
      dim: 0
      dim: -1
    }
  }
}
layer {
  name: "conv3_1_axpy_scale"
  type: "Scale"
  bottom: "conv3_1_1x1_increase"
  bottom: "conv3_1_axpy_reshape"
  top: "conv3_1_axpy_scale"
  scale_param {
    axis: 0
    bias_term: false
```

```
  }
}
layer {
  name: "conv3_1_axpy_eltwise"
  type: "Eltwise"
  bottom: "conv3_1_axpy_scale"
  bottom: "conv3_1_1x1_proj"
  top: "conv3_1"
}
```

Reshape、Scale、Eltwise算子相关参数解释请参见**15.3.1 Caffe框架算子规格**。

15.5 使用 AIPP 色域转换模型时如何判断视频流的格式标准

Q：使用AIPP色域转换模型时如何判断视频流的格式标准？

A：这里以第三方**ffprobe**工具为例，您也可以使用其他第三方工具。

1. 您可以从官方下载工具及相关文档，**https://www.ffmpeg.org/ffprobe-all.html#Description**。
2. 通过 **ffprobe -show_frames** ***filename*** 参数获取对应视频信息。

   参数功能： Show information about each frame and subtitle contained in the input multimedia stream. The information for each single frame is printed within a dedicated section with name "FRAME" or "SUBTITLE".
3. 通过结果中如下信息确认是哪种视频标准。

   "color_range": "tv" 或者 "pc"

   "color_space": "bt709" 或者 "bt601"

   其中tv代表：tv表示 limited，即narrow range。pc代表：pc表示full，即wide range。

   例如查询结果为color_range=tv，color_space=bt709，则代表BT-709，NARROW。

   说明

   如命令变化，请以工具官方说明为准。

15.6 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：

```
export http_proxy="http://user:password@proxyserverip:port"
export https_proxy="http://user:password@proxyserverip:port"
```

其中user为用户在内部网络中的用户名，password为用户密码，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。

----结束

15.7 安全加固建议

- 63342~63391端口段是MindStudio的内置Server服务端口段，可用于打开本地的web服务、html文件和xml文件等。为了提高安全性，建议此端口段仅支持本地使用，在防火墙中不要对其他设备开放。
- 6942~6991端口段是MindStudio进程单实例绑定端口段，用于防止启动多个IDE进程，并不用于通信。为了提高安全性，建议此端口段仅支持本地使用，在防火墙中不要对其他设备开放。
- 在应用工程执行远程运行等操作时，会把相关文件传输到远端，为了提高安全性，建议远端用户的umask值设置为0027。

15.8 FAQ

15.8.1 Windows 上远程打开 MindStudio 时，偶现复制的内容无法粘贴到编辑器窗口中

问题现象

Windows上远程打开MindStudio时，偶现复制的内容无法粘贴到编辑器窗口中。

解决方案

MobaXterm自带划取复制能力，导致打开的MindStudio也带上该能力，可以设置disable “select on copy”来解决，如下图所示。

也可以当复制内容之后，在MindStudio编辑器窗口中光标不要再次选中内容，直接将复制的内容粘贴到对应位置即可解决。

15.8.2 MobaXterm 启动 MindStudio 编辑字符时，偶现方向键变成数字

问题现象

MobaXterm启动MindStudio编辑代码或者在MindStudio配置界面输入字符时，偶现方向键变成2468的数字。

解决方案

卸载或者切换不使用搜狗输入法，然后用MobaXterm远程连接启动MindStudio。

15.8.3 配置不受信任的网址访问浏览器

UT用例运行结束后，如果需要通过浏览器打开覆盖率html文件，可能会出现图15-9所示提示。

图 15-9 提示

请单击菜单栏中的“File > Settings”，参考图15-10修改。

图 15-10 勾选 Allow unsigned requests

15.8.4 启动 MindStudio 报错：Failed to initialize graphics environment

问题

启动MindStudio时，报错Failed to initialize graphics environment，如图15-11所示。

图 15-11 GUI 初始化失败

```
Please use Python interpreter parameter [ /usr/local/python3.7.5/bin/python3 ] to configure Python SDK.
MobaXterm X11 proxy: Authorisation not recognised
2020-12-12 10:32:22,153 [      32]  ERROR - llij.ide.plugins.PluginManager - UI initialization failed
com.intellij.ide.plugins.StartupAbortedException: UI initialization failed
        at com.intellij.idea.StartupUtil.lambda$prepareApp$2(StartupUtil.java:188)
        at java.util.concurrent.CompletableFuture.uniExceptionally(CompletableFuture.java:884)
        at java.util.concurrent.CompletableFuture$UniExceptionally.tryFire(CompletableFuture.java:866)
        at java.util.concurrent.CompletableFuture.postComplete(CompletableFuture.java:488)
        at java.util.concurrent.CompletableFuture.completeExceptionally(CompletableFuture.java:1990)
        at com.intellij.idea.StartupUtil.lambda$scheduleInitUi$9(StartupUtil.java:372)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.util.concurrent.CompletionException: java.awt.AWTError: Can't connect to X11 window server using 'localhost:15.0' as the value of th
        at java.util.concurrent.CompletableFuture.encodeThrowable(CompletableFuture.java:292)
        at java.util.concurrent.CompletableFuture.completeThrowable(CompletableFuture.java:308)
        at java.util.concurrent.CompletableFuture.biRelay(CompletableFuture.java:1298)
        at java.util.concurrent.CompletableFuture$BiRelay.tryFire(CompletableFuture.java:1284)
        ... 6 more
Caused by: java.awt.AWTError: Can't connect to X11 window server using 'localhost:15.0' as the value of the DISPLAY variable.
        at sun.awt.X11GraphicsEnvironment.initDisplay(Native Method)
        at sun.awt.X11GraphicsEnvironment.access$200(X11GraphicsEnvironment.java:65)
        at sun.awt.X11GraphicsEnvironment$1.run(X11GraphicsEnvironment.java:115)
        at java.security.AccessController.doPrivileged(Native Method)
        at sun.awt.X11GraphicsEnvironment.<clinit>(X11GraphicsEnvironment.java:74)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:264)
        at java.awt.GraphicsEnvironment.createGE(GraphicsEnvironment.java:103)
        at java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment(GraphicsEnvironment.java:82)
        at sun.awt.X11.XToolkit.<clinit>(XToolkit.java:131)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:264)
        at java.awt.Toolkit$2.run(Toolkit.java:860)
        at java.awt.Toolkit$2.run(Toolkit.java:855)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.awt.Toolkit.getDefaultToolkit(Toolkit.java:854)
        at java.awt.Toolkit.getEventQueue(Toolkit.java:1736)
        at java.awt.EventQueue.invokeLater(EventQueue.java:1294)
        at com.intellij.idea.StartupUtil.lambda$scheduleInitUi$9(StartupUtil.java:323)
        ... 3 more
2020-12-12 10:32:22,155 [      34]  ERROR - llij.ide.plugins.PluginManager - MindStudio 2.0.0  Build #MS-201.8743.12.3
2020-12-12 10:32:22,155 [      34]  ERROR - llij.ide.plugins.PluginManager - JDK: 1.8.0_275; VM: OpenJDK 64-Bit Server VM; Vendor: Private Build
2020-12-12 10:32:22,156 [      35]  ERROR - llij.ide.plugins.PluginManager - OS: Linux

Start Failed: Failed to initialize graphics environment
```

原因

- 如果安装的是服务器版的ubuntu系统，使用某用户登录了MobaXterm，所以图形工具已经被该用户所在的用户组占用，切换用户后，无法使用图形工具。
- 如果安装的是桌面版的ubuntu系统，登录ubuntu系统的用户和启动MindStudio用户不一致。

解决方法

- 如果安装的是服务器版的ubuntu系统，请参见以下方法解决：
  - 把切换后的用户加入先前用户的用户组。
  - 使用想要启动MindStudio的用户重新登录MobaXterm。
  - 在当前启动MindStudio的用户下设置DISPLAY环境变量：
    export DISPLAY=*显示图形界面的XServer主机IP:0*
  - 在当前启动MindStudio的用户下进入**.bashrc**，添加**export DISPLAY=*显示图形界面的XServer主机IP:0***环境变量后保存退出，然后执行**source ~/.bashrc**即可。
- 如果安装的是桌面版的ubuntu系统，请参见以下方法解决：
  - 使用启动MindStudio用户重新登录ubuntu系统。

15.8.5 TensorFlow 导出 Pb 模型

使用Keras构建模型的用户，可尝试如下方法进行导出。

- 对于TensorFlow 1.15.x版本：

```
import tensorflow as tf
from tensorflow.python.framework import graph_io
from tensorflow.python.keras.applications.inception_v3 import InceptionV3


def freeze_graph(graph, session, output_nodes, output_folder: str):
    """
    Freeze graph for tf 1.x.x.
```

```
    Args:
        graph (tf.Graph): Graph instance.
        session (tf.Session): Session instance.
        output_nodes (list): Output nodes name.
        output_folder (str): Output folder path for frozen model.

    """
    with graph.as_default():
        graphdef_inf = tf.graph_util.remove_training_nodes(graph.as_graph_def())
        graphdef_frozen = tf.graph_util.convert_variables_to_constants(session, graphdef_inf,
output_nodes)
        graph_io.write_graph(graphdef_frozen, output_folder, "frozen_model.pb", as_text=False)


tf.keras.backend.set_learning_phase(0)

keras_model = InceptionV3()
session = tf.keras.backend.get_session()

INPUT_NODES = [ipt.op.name for ipt in keras_model.inputs]
OUTPUT_NODES = [opt.op.name for opt in keras_model.outputs]
freeze_graph(session.graph, session, OUTPUT_NODES, "/home/user/xxx")
print(f"Input nodes name: {INPUT_NODES}, output nodes name: {OUTPUT_NODES}")
```

- 对于TensorFlow 2.x.x版本：

```
import tensorflow as tf
from tensorflow.python.framework.convert_to_constants import convert_variables_to_constants_v2


def convert_to_froze_graph(keras_model: tf.python.keras.models.Model, model_name: str,
                           output_folder: str):
    """
    Export keras model to frozen model.

    Args:
        keras_model (tensorflow.python.keras.models.Model):
        model_name (str): Model name for the file name.
        output_folder (str): Output folder for saving model.

    """
    full_model = tf.function(lambda x: keras_model(x))
    full_model = full_model.get_concrete_function(
        tf.TensorSpec(keras_model.inputs[0].shape, keras_model.inputs[0].dtype)
    )

    frozen_func = convert_variables_to_constants_v2(full_model)
    frozen_func.graph.as_graph_def()

    print(f"Model inputs: {frozen_func.inputs}")
    print(f"Model outputs: {frozen_func.outputs}")

    tf.io.write_graph(graph_or_graph_def=frozen_func.graph,
                      logdir=output_folder,
                      name=model_name,
                      as_text=False)
```

15.8.6 操作 MindStudio 时弹出 Unlock Keyring 窗口：Enter password to unlock

问题现象

在操作MindStudio时弹出需要输入密码的窗口，如下图所示。

解决方法

输入SSH加密密码，单击“Unlock”即可。SSH加密密码相关操作可参见**13.1.6 加密SSH密码**。

15.8.7 如何获取帮助和提出建议

如果在使用MindStudio的过程中，有任何问题想要咨询或者提出改进建议，请直接在MindStudio主界面的菜单栏中单击“Help > Ascend Developer Zone”，如**图15-12**所示，进入昇腾论坛进行反馈。

说明

使用Ascend Developer Zone功能时需要使用浏览器，如果未安装浏览器，请用户自行安装。

图 15-12 Ascend Developer Zone

15.9 修订记录

表 15-3 修订记录

发布日期	修订记录
2021-01-22	第一次正式发布。