MindStudio 6.0.0 用户指南

文档版本	01
发布日期	2025-02-12

华为技术有限公司

商标声明


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

注意

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

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

安全声明

产品生命周期政策

华为公司对产品生命周期的规定以“产品生命周期终止政策”为准，该政策的详细内容请参见如下网址：
https://support.huawei.com/ecolumnsweb/zh/warranty-policy

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

华为初始证书权责说明

华为公司对随设备出厂的初始数字证书，发布了“华为设备初始数字证书权责说明”，该说明的详细内容请参见如下网址：
https://support.huawei.com/enterprise/zh/bulletins-service/ENEWS2000015766

华为企业业务最终用户许可协议(EULA)

本最终用户许可协议是最终用户（个人、公司或其他任何实体）与华为公司就华为软件的使用所缔结的协议。最终用户对华为软件的使用受本协议约束，该协议的详细内容请参见如下网址：
https://e.huawei.com/cn/about/eula

产品资料生命周期策略

华为公司针对随产品版本发布的售后客户资料（产品资料），发布了“产品资料生命周期策略”，该策略的详细内容请参见如下网址：
https://support.huawei.com/enterprise/zh/bulletins-website/ENEWS2000017760

目 录

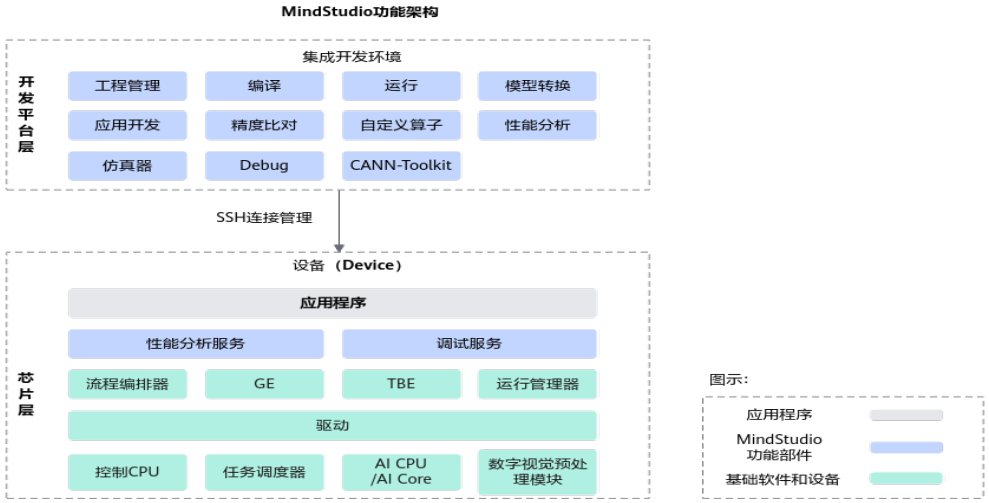
1 认识 MindStudio

1.1 功能架构

MindStudio提供您在AI开发所需的一站式开发环境，支持模型开发、应用开发以及算子开发三个主流程中的开发任务。依靠模型可视化、算力测试、IDE本地仿真调试等功能，MindStudio能够帮助您在一个工具上就能高效便捷地完成AI应用开发。

MindStudio功能框架如图1-1所示。

图 1-1 工具链功能架构



说明

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_64服务器、aarch64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之相连接的ARM服务器合设，统称为Host。

1.2 关键功能概览

工程管理

MindStudio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。

推理一体化工具

模型推理一体化工具AMIT（Ascend Model Inference Tool），支持用户在统一入口下调用模型分析、模型转换、离线模型dump、精度比对和性能分析工具，提升开发效率。同时提供步骤指引，方便用户理解推理开发流程，更易上手使用。

分析迁移

- X2MindSpore工具：将PyTorch/TensorFlow训练脚本迁移至可基于MindSpore运行的代码。
- PyTorch GPU2Ascend工具：分析PyTorch训练脚本的算子支持情况，支持将PyTorch训练脚本从GPU平台迁移至昇腾NPU平台。

模型训练

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，对网络分析结果进行界面展示。

模型转换

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。

应用开发

基于MindSDK开发应用：MindSDK采用模块化设计理念，将业务流程中的各个功能单元封装成独立的插件，通过插件串联，快速构建业务。

MindStudio基于MindSDK，提供可视化流程编排功能，通过画板拖动插件的方式，可视直观地组织插件，管理整体流程，最终生成pipline文件以供开发。

基于AscendCL开发应用：通过MindStudio，开发基于AscendCL（Ascend Computing Language）应用工程，利用AscendCL提供的C语言API库，对昇腾硬件计算资源进行

管理和调配，在昇腾CANN平台上进行深度学习推理计算、图形图像预处理、单算子加速计算等工作。

算子开发

MindStudio提供包含UT测试、ST测试、TIK算子调试等的全套算子开发流程。支持TensorFlow、PyTorch、MindSpore等多种主流框架的TBE和AI CPU算子开发。

- UT测试：UT（Unit Test）即单元测试，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。
- ST测试：ST（System Test）即系统测试，MindStudio提供了新的ST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。
- TIK算子调试：MindStudio支持TIK算子的可视化调试，可以实现断点设置、单步调试、连续运行直到结束或下一断点、查看变量信息、退出调试等功能。

精度比对

精度比对工具是为了帮助开发人员快速解决算子精度问题，提供自有实现的算子运算结果与业界标准算子运算结果之间进行精度差异对比的工具。

性能分析

性能分析工具提供了AI应用运行过程中软、硬件相关性能数据的采集和性能指标的分析，并通过可视化界面的方式呈现，可以帮助用户快速发现和定位AI应用的性能瓶颈，显著提升AI任务性能分析的效率。

专家系统

专家系统（MindStudio Advisor）是用于聚焦模型和算子的性能调优Top问题，识别性能瓶颈，重点构建瓶颈分析、优化推荐模型，支撑开发效率提升的工具。

算子和模型速查

算子和模型速查工具帮助客户了解当前版本CANN支持的算子信息和ModelZoo支持的模型信息。

2 工程管理

2.1 工程创建

2.1.1 昇腾工程

须知

不建议用户将较大的数据文件置于工程目录下，过大的工程目录文件可能会导致工程在打开以及编写过程中，出现迟缓、卡顿等情况。

MindStudio支持创建以下类型的昇腾工程，详细信息请参见以下对应章节。

- **模型训练工程**
- **应用开发工程**
- **算子开发工程**

2.1.2 非昇腾工程

2.1.2.1 简介

须知

不建议用户将较大的数据文件置于工程目录下，过大的工程目录文件可能会导致工程在打开以及编写过程中，出现迟缓、卡顿等情况。

MindStudio支持创建和导入纯语言类工程、Empty工程。

2.1.2.2 创建 C/C++工程

MindStudio支持创建“C Executable”、“C Library”、“C++ Executable”和“C++ Library”四种类型的C/C++工程，创建方式相似（以下步骤以创建“C++ Executable”为例）。

说明

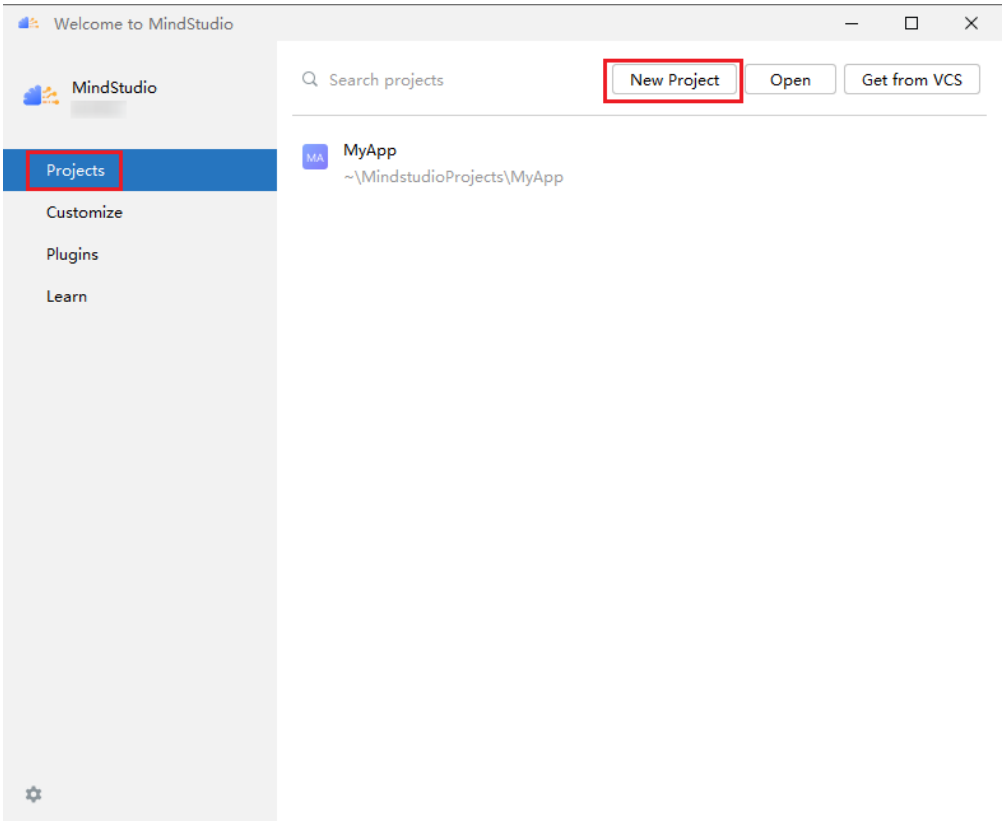
- 仅支持创建CMake编译框架的C/C++工程。
- C标准支持范围为C90、C99、C11，C++标准支持范围为C++98、C++11、C++14、C++17、C++20。
- Library工程支持**static**和**shared**两种类型。
- MindStudio支持对.c、.cpp、.cc、.h源文件关键字着色、跳转功能。

新建 C/C++工程方式

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

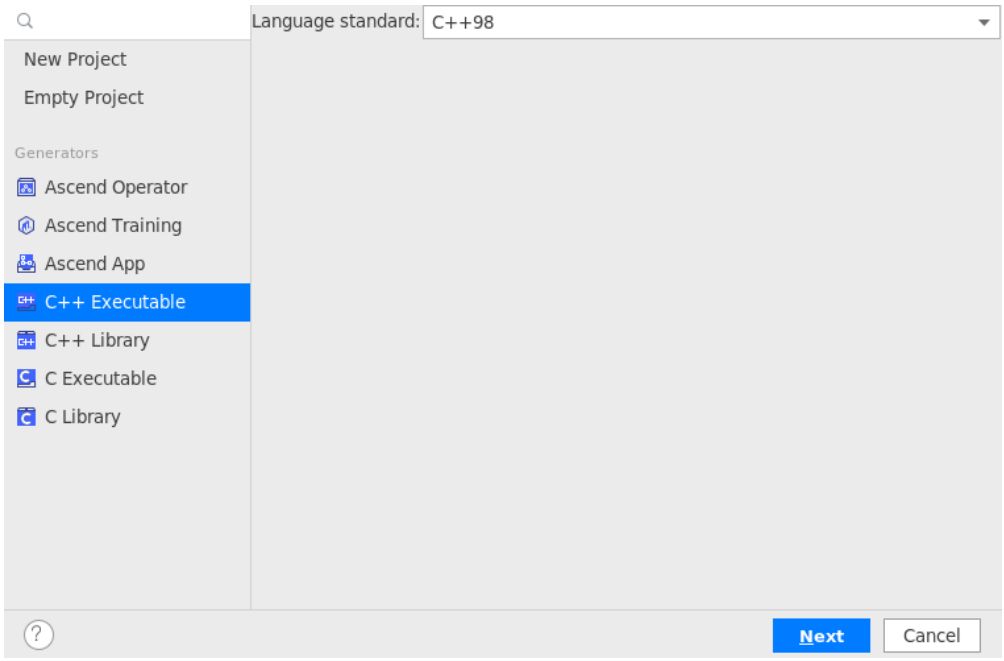
图 2-1 工程创建界面



- MindStudio工程界面：在顶部菜单栏中选择“File > New > Project...”。

步骤2 在**New Project**窗口中，左侧菜单选择“C++ Executable”，在右侧窗口中选择“Language standard”（C++标准支持范围，默认“C++ 98”。），单击“Next”进入下一步。

图 2-2 选择“C++ Executable”



步骤3 进入工程配置栏，配置工程相关参数如表2-1所示。

表 2-1 工程参数说明

参数	说明
Project name	工程名称，自行配置。 名称开头和结尾必须是数字或字母。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。
Project location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）
More Settings	“Module name”：模块名，默认与“Project name”一致。
	“Content root”：根目录下路径。
	“Module file location”：模块文件路径。
	单击“Project format”右侧选框，出现下拉菜单。 • .idea（directory-based）：创建项目的时候创建一个.idea项来保存项目的信息，默认选项。 • .ipr（file-based）：项目配置文件来保存项目的配置信息。

步骤4 单击“Create”，完成工程创建。

若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤5 成功创建工程后，工程目录以树状呈现。

----结束

导入 C/C++工程方式

步骤1 导入工程文件。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的  ，选择现有工程打开。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 选择“This Window”，则直接在当前工作窗口打开工程。
- 选择“New Window”，则新建一个工作窗口并打开工程。

步骤3 成功导入工程后，工程目录以树状呈现。

----结束

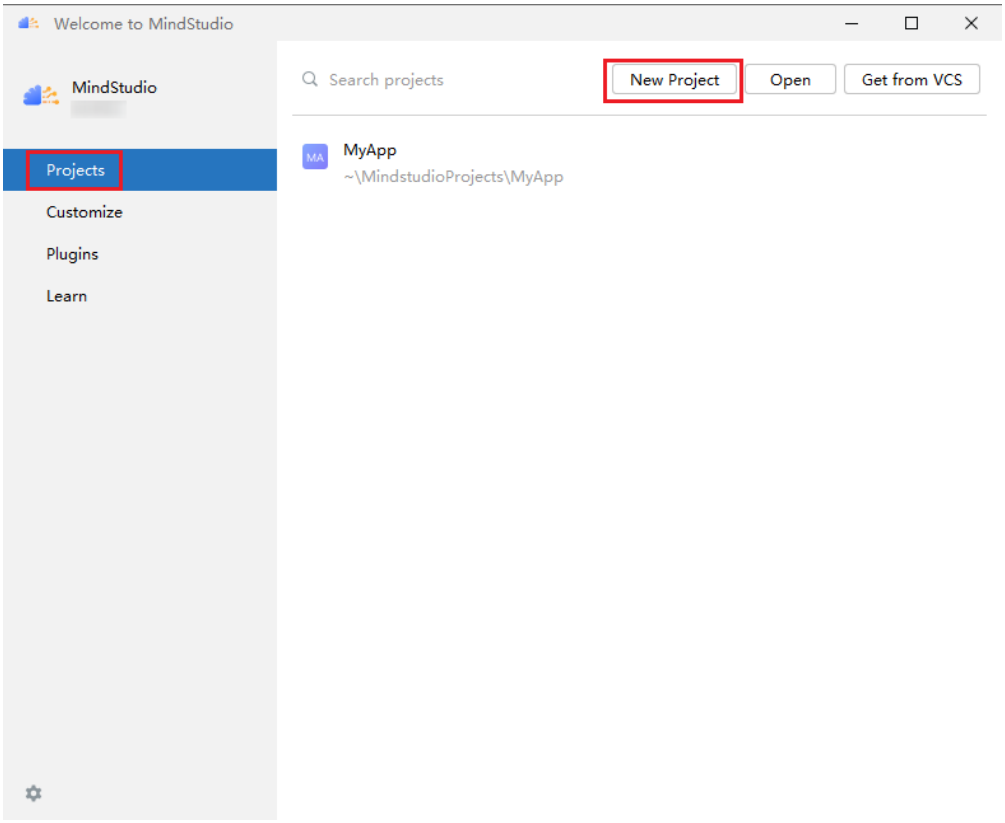
2.1.2.3 创建 Java 工程

新建 Java 工程方式

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

图 2-3 工程创建界面



- MindStudio工程界面：在顶部菜单栏中选择“File > New > Project...”。

步骤2 在“New Project”窗口中，左侧菜单选择“New Project”，配置工程相关参数如**表1 参数说明**所示。

表 2-2 参数说明

参数	说明
Name	工程名称，用户可自行配置。
Location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。） Create Git repository：是否创建Git仓库，勾选后则创建。
Language	可选择“Java”、“Python”或点击右侧“+”选择更多。创建Java工程时选择“Java”。
Build system	选择构建系统，可选择“IntelliJ”或“Gradle”。 若选择“Gradle”，会新增“Gradle DSL”选项，当前可选择“Groovy”或“Kotlin”。
JDK	选择JDK。点击右侧下拉框可选择： • Download JDK：下载JDK。 • Add JDK：从本地目录添加JDK。

参数	说明
Add sample code	是否添加样例代码。勾选后则会在“src”目录下创建样例代码。
Advanced Settings	“Module name”：模块名，默认与“Name”一致。
	“Content root”：根目录下路径。
	“Module file location”：模块文件路径。

步骤3 单击“Create”，完成工程创建。

若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤4 成功创建工程后，工程目录以树状呈现，请以实际创建结果为准。

说明

用户也可以通过顶部菜单栏进入“File > Project Structure…”，在弹出的左侧功能栏中选中“SDKs”并在右侧功能界面单击“+”，选择“Download JDK”，自行下载Java工程的SDK。

（JDK默认访问地址https://download-cdn.jetbrains.com/jdk/feed/v1/jdks.json.xz ）

----结束

导入 Java 工程方式

步骤1 导入工程文件。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的  ，选择现有工程打开。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in <*工作区目录*>”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 选择“This Window”，则直接在当前工作窗口打开工程。
- 选择“New Window”，则新建一个工作窗口并打开工程。

步骤3 成功导入工程后，工程目录以树状呈现。

----结束

2.1.2.4 创建 Python 工程

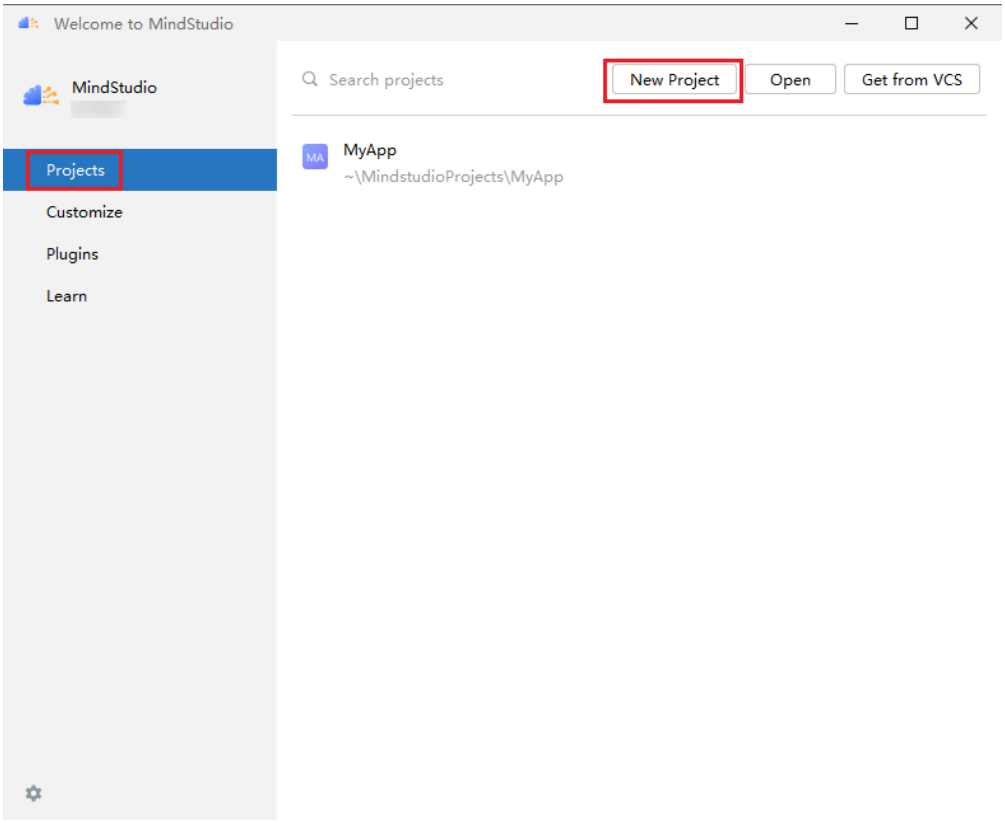
MindStudio支持创建Python工程，进行Python工程的开发调测，并支持远端进行调测的能力。

新建 Python 工程方式

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

图 2-4 工程创建界面



- MindStudio工程界面：在顶部菜单栏中选择“File > New > Project...”。

步骤2 在“New Project”窗口中，左侧菜单选择“New Project”，配置工程相关参数如**表2-3**所示。

表 2-3 工程参数说明

参数	说明
Name	工程名称，用户可自行配置。
Location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。） Create Git repository：是否创建Git仓库，勾选后则创建。

参数	说明
Language	可选择“Java”、“Python”或点击右侧“+”选择更多。创建Python工程时选择“Python”。
Environment	选择Python环境，可选择“New”或“Existing”。 选择“New”后可新增环境类型，选择“Existing”后可选择已经存在的解析器。
Environment type	选择环境类型。当前可选择“Virtualenv”、“Conda”、“Pipenv”或“Poetry”。
Location	“Environment type”选择“Virtualenv”或“Conda”会出现该参数。 新的“Virtualenv”或“Conda”环境的位置。用户可以点击右侧文件夹图标选择所需的位置。
Base interpreter	“Environment type”选择“Virtualenv”、“Pipenv”或“Poetry”会出现该参数。 用户可以从下拉列表中选择基本解析器，或者点击右侧“...”选择所需的Python可执行文件。
Inherit global site-packages	“Environment type”选择“Virtualenv”会出现该参数。 勾选该选项后，计算机上安装在全局Python中的所有软件包将添加到要创建的虚拟环境中。
Make available to all projects	“Environment type”选择“Virtualenv”或“Conda”会出现该参数。 勾选该选项后，MindStudio创建Python解析器时会重用此环境。
Python version	“Environment type”选择“Conda”会出现该参数。 点击下拉框选择Python版本。
Conda executable	“Environment type”选择“Conda”会出现该参数。 Conda可执行文件的位置。系统将自动检测Conda可执行文件的路径，用户可单击右侧文件夹图标选择Conda可执行文件路径。
Pipenv executable	“Environment type”选择“Pipenv”会出现该参数。 Pipenv可执行文件的位置。系统将自动检测Pipenv可执行文件的路径，用户可单击右侧文件夹图标选择Pipenv可执行文件路径。
Poetry executable	“Environment type”选择“Poetry”会出现该参数。 Poetry可执行文件的位置。系统将自动检测Poetry可执行文件的路径，用户可单击右侧文件夹图标选择Poetry可执行文件路径。

步骤3 单击“Create”，完成工程创建。

若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤4 成功创建工程后，工程目录以树状呈现。

----结束

导入 Python 工程方式

步骤1 导入工程文件。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的  ，选择现有工程打开。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 选择“This Window”，则直接在当前工作窗口打开工程。
- 选择“New Window”，则新建一个工作窗口并打开工程。

步骤3 成功导入工程后，工程目录以树状呈现。

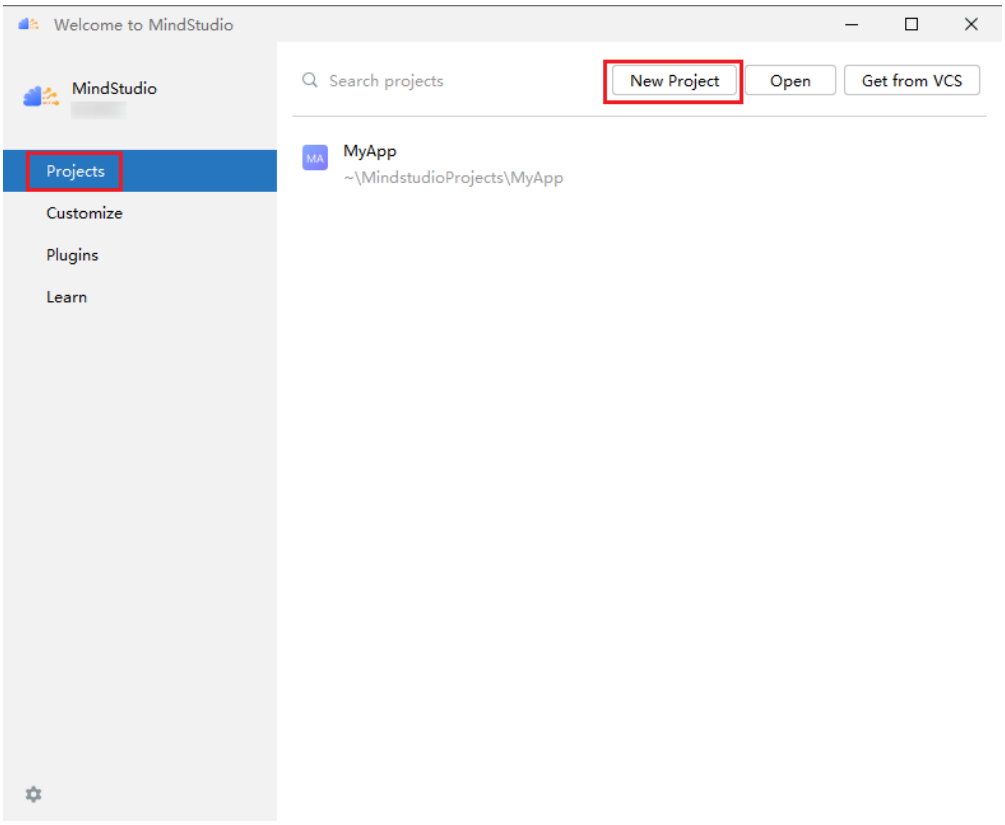
----结束

2.1.2.5 创建 Empty 工程

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

图 2-5 工程创建界面



- MindStudio工程界面：在顶部菜单栏中选择“File > New > Project...”。

步骤2 在“New Project”窗口中，左侧菜单选择“Empty Project”。

步骤3 单击“Next”，进入配置工程相关参数如表2-4所示。

表 2-4 工程参数说明

参数	说明
Name	工程名称，自行配置。
Location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）
Create Git repository	是否创建Git仓库，勾选后则创建。

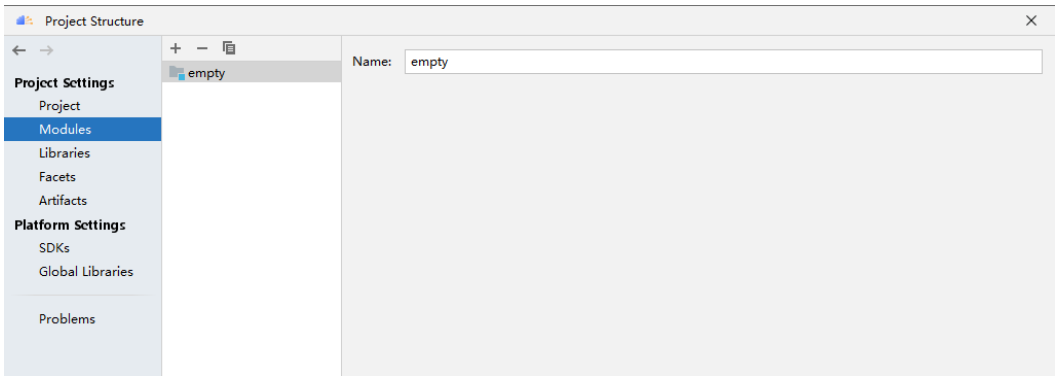
步骤4 单击“Create”，完成工程创建。

若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤5 成功创建工程后，可通过顶部菜单栏找到“File > Project Structure... ”打开如图2-6所示设置窗口，通过左侧功能栏中的“Modules”功能，手动创建或导入**Module**模块。

图 2-6 Project Structure 设置窗口



----结束

2.1.3 工程转换

MindStudio支持非昇腾工程转换为昇腾工程的功能。

操作步骤

步骤1 导入工程文件。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的，选择现有工程打开。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

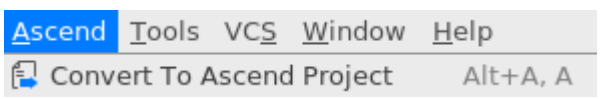
- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 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。

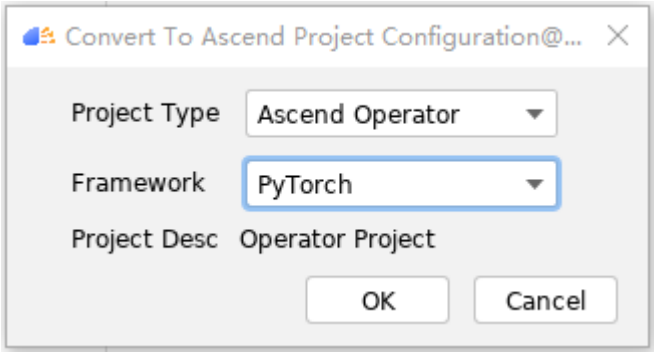
步骤3 单击顶部菜单栏中的“Ascend > Convert To Ascend Project”，如图2-7所示。

图 2-7 工程转换



步骤4 弹出如图2-8窗口。

图 2-8 转换昇腾工程配置



对窗口参数介绍如下，用户请根据实际场景选择：

- **Project Type**：可选工程类型分别为“Ascend Operator”、“Ascend Training”、“Ascend App”。
- **Framework/Sub Type**：可选择框架如表2-5、表2-6所示。

表 2-5 FrameWork 可选框架

Project Type	Framework
Ascend Operator	MindSpore
	PyTorch
	TensorFlow
	ONNX
Ascend Training	MindSpore
	PyTorch
	TensorFlow

表 2-6 Sub Type 可选子类型

Project Type	Sub Type
Ascend App	Ascend ACL App
	Ascend Python ACL App
	Ascend MindSDK App

- **Project Desc**：项目描述。

步骤5 单击“OK”，工程目录以树状呈现。此时成功创建带有.project文件的昇腾工程，请以实际创建结果为准。

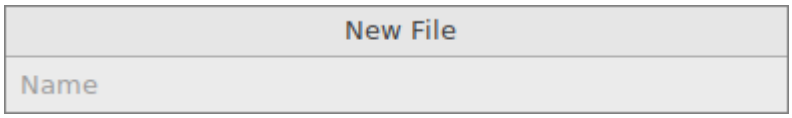
----结束

2.2 工程文件管理

2.2.1 添加工程文件

在工程界面下，选中相应工程，单击鼠标右键选择“New”（或者在顶部菜单栏中选择“File > New”），在弹出的子菜单中选择需要创建的文件类型或选择“File”，如图2-9所示，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）。

图 2-9 输入新建文件名



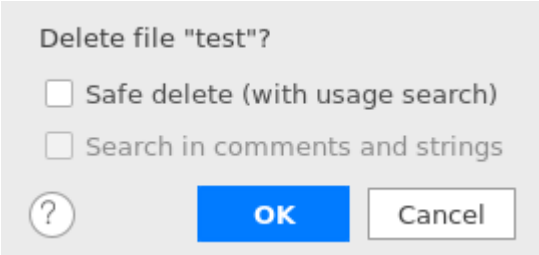
在输入文件名后单击“回车”即可完成创建。

2.2.2 删除工程文件

在左侧工程目录中选中文件，单击鼠标右键，在弹出的菜单中单击“Delete...”，提示删除选项，如图2-10所示，在删除选项窗口中可选择以下方式进行删除。

- Safe delete（使用安全删除方案）：将通过搜索使用情况执行安全删除，确保该文件在删除时没有被引用，如搜索出待确认的情况，MindStudio会展示整体搜索结果以及具体使用情况，用户可根据结果分别处理。
- （可选）Search in comments and strings（搜索注释和字符串）：追加搜索注释与字符串的引用情况。

图 2-10 删除文件



2.3 工程调试与执行

2.3.1 调试 C/C++代码

2.3.1.1 调试前准备

说明

支持在Linux环境与Windows环境进行应用工程调试，并具备远程调试的功能。

GDB 安装

须知

以下步骤仅供参考，具体安装步骤请参见**GDB官方文档**。

若环境中未安装GDB，则需要安装GDB，可通过包管理（如**apt-get install gdb**、**yum install gdb**）进行安装。

如无法使用包管理安装GDB，则需通过源码编译方式安装GDB，操作步骤如下：

步骤1 获取GDB源码（**获取链接**），如获取“gdb-8.1.1.tar.gz”。

步骤2 解压GDB源码压缩包，以“gdb-8.1.1.tar.gz”为例。

```
tar -zxvf gdb-8.1.1.tar.gz
```

步骤3 进入到GDB目录文件，配置编译参数：

```
cd gdb-8.1.1
./configure --host=aarch64-linux-gnu --prefix=/home/install/gdb
```

- “--host”为可选参数，在需要交叉编译安装GDB时使用。“aarch64-linux-gnu”为用户获取的交叉编译器示例。
- “/home/install/gdb”为指定安装目录示例，请用户根据实际情况自行替换。

步骤4 编译。

```
make -j8
make install
```

步骤5 配置环境变量。

```
export PATH=/home/install/gdb/bin:$PATH
```

步骤6 查看GDB是否安装成功。

```
gdb --version
```

如出现类似如下回显，表明安装成功。

```
GNU gdb (GDB) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
```

----结束

2.3.1.2 编译与运行

步骤1 在菜单栏选择“Build > Edit Configurations...”进入编译配置界面，单击“+”按钮，添加配置选项如**图2-11**所示。（列表中带有“(default)”标记的配置项为当前默认配置项，如需进行变更，可选中所需配置项，并单击 进行切换。）

图 2-11 Build Configurations 配置

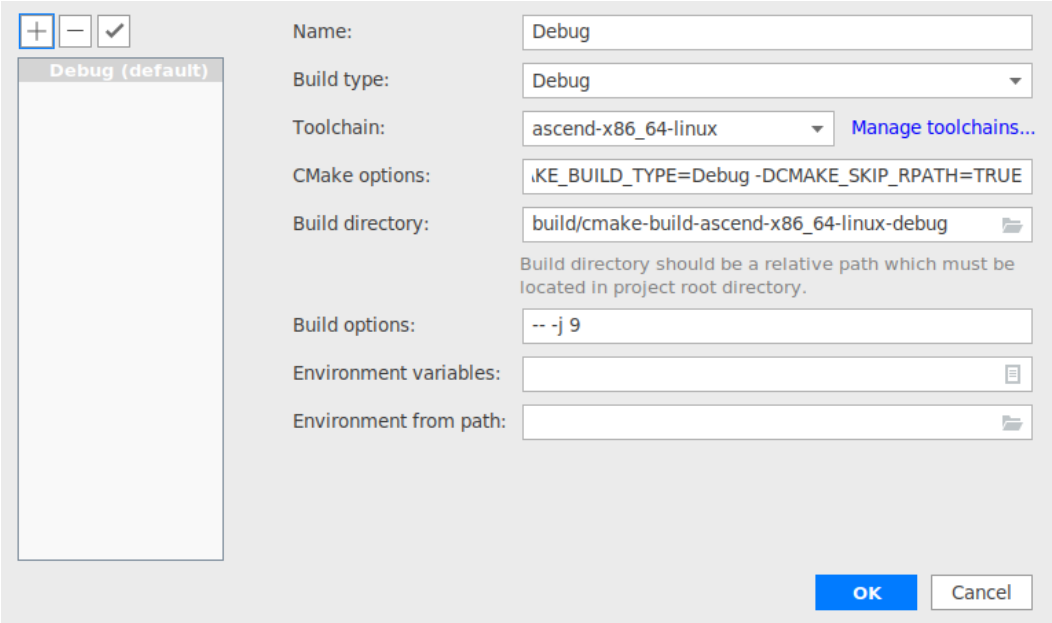


表 2-7 Edit Configurations 配置说明

参数及图标	说明
Name	配置名称自定义，默认：“Debug”。
Build type	配置类型，可选：“Debug”或“Release”。
Toolchain	工具链配置器，配置详情请参见13.4 Toolchains，支持本地和远程编译功能。
CMake options	CMake选项，默认：“-DCMAKE_BUILD_TYPE=Debug -DCMAKE_SKIP_RPATH=TRUE”。
Build directory	编译目录相对路径，该路径是相对于工程目录的路径。
Build options	编译加速选项。
Environment variables	环境变量配置：支持编译前配置环境变量。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

步骤2 请用户根据实际使用场景，在MindStudio工程界面，选择以下编译方式，如需切换默认配置项请参见步骤1。

- 依次选择“Build > Clean CMake Project”，清理编译。
- 依次选择“Build > Rebuild CMake Project”，使用默认配置进行全量编译。
- 依次选择“Build > Build CMake Project”或单击工具栏中的，使用默认配置进行增量编译。

步骤3 在顶部菜单栏中选择“Run > Edit Configurations...”或通过单击工具栏中的运行配置框进入运行配置，单击左上角“+”，新建“CMake Application”运行配置，如图2-13所示，运行配置项说明请参见表2-8。

单击“OK”按钮，结束编译与运行配置准备。

图 2-12 运行配置框

图 2-13 运行 Edit Configurations 配置

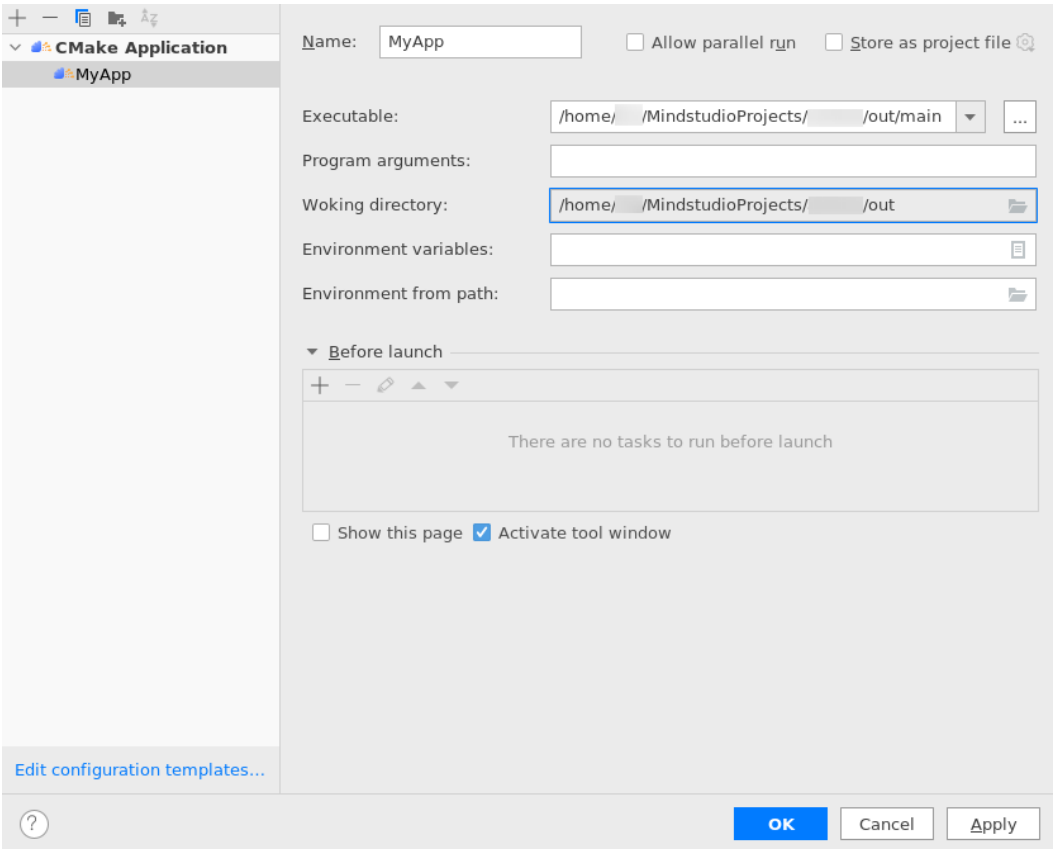


表 2-8 运行配置项

配置项	说明
Executable	选择编译用可执行文件的路径，选择到文件一级。
Program arguments	运行参数。
Working directory	工作目录。（如工程中配置了相对路径，则需在该选项中选择相对的工作目录。默认为可执行文件所在目录。）
Enviroment variables	此处直接输入动态链接库路径。 或者单击，在弹出的界面内单击，填写路径。

配置项	说明
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

步骤4 在顶部菜单栏中选择“Run > Run '*工程名*”，使用已保存的配置运行工程。

如未保存运行配置，在当前操作后，会弹出运行配置界面，运行配置填写请参见**步骤3**。填写完成，可单击“Apply”应用配置后，单击“Run”直接运行。

----结束

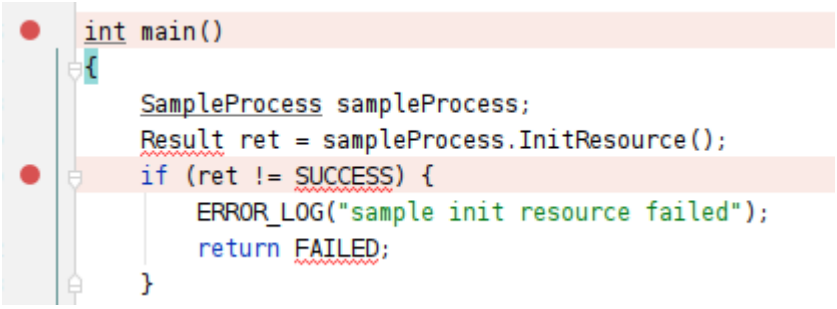
2.3.1.3 断点管理

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，然后用户可通过预先完成的设置，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，即可以添加断点，如**图2-14**所示，断点添加成功后，左侧代码行号处会显示图标。

图 2-14 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。

停用断点

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，则可以通过以下操作暂时停用断点。

右击已经设置断点的代码行号处的，在弹出界面中不勾选“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。

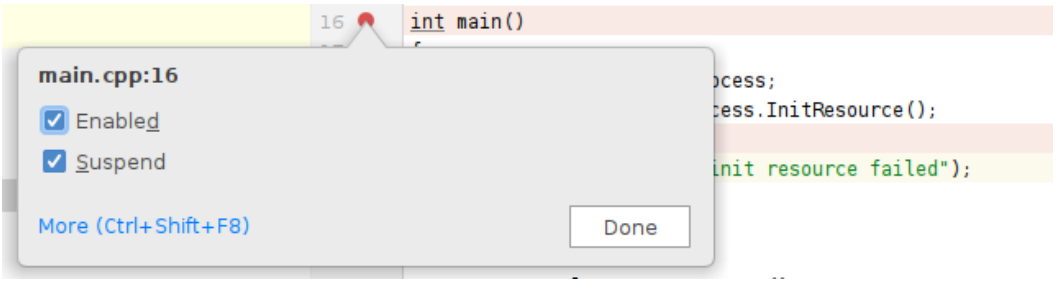
图 2-15 断点停用

启用断点

若已经有部分断点设置了停用，执行新的调试功能时想重新启用断点，则可以通过以下操作实现。

右击已经停用断点处的，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 2-16 断点启用



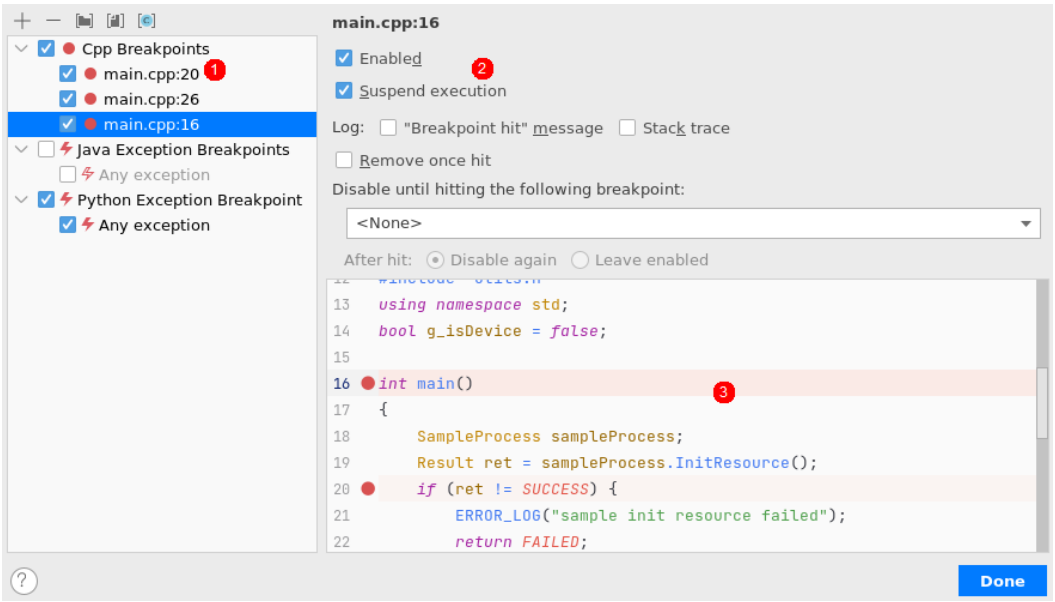
说明

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

在顶部菜单栏中找到“Run > View Breakpoints”或单击断点界面的“More(Ctrl+Shift+F8)”，弹出图2-17所示断点查看界面。通过该功能，用户可以查看当前设置的所有断点。

图 2-17 断点查看界面



左侧序号1断点列表展示了当前代码中所设置的所有断点，序号2对应相应断点的停用（Disabled）或启用（Enabled）功能，序号3为代码预览区域。

功能使用方法参考如下：

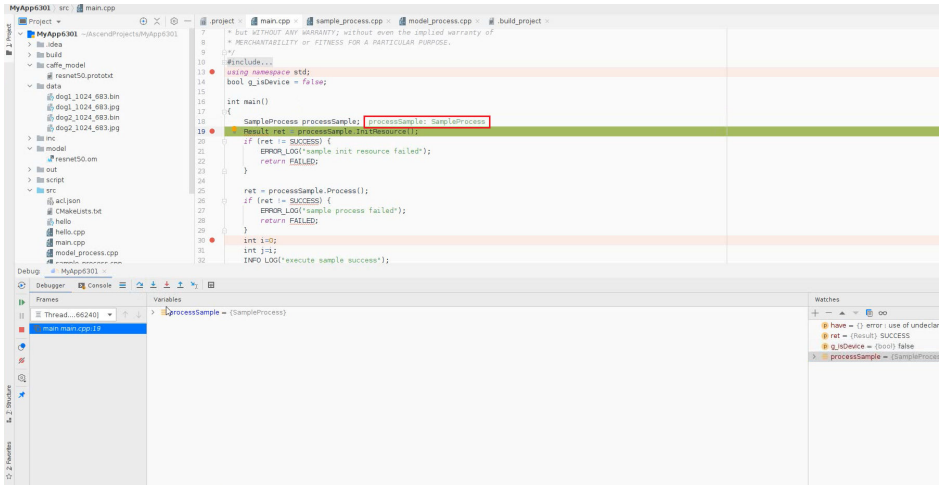
- 单击左侧断点列表中，断点左侧的复选框，勾选表示该断点启用，相应序号2处“Enabled”功能被勾选，同时在代码预览区域中，对应代码行左侧行号处显示 图标；取消勾选表示该断点停用，相应序号2处“Enable”功能被取消勾选，同时在代码预览区域中，对应代码行左侧行号处显示 图标。
- 选中断点列表处的断点，单击上方的 可以删除断点，同时代码预览区域，相应代码行号处的断点图标随之删除。
- 在代码预览区域，代码行号处单击，则该行被标记为断点，同时断点列表中会自动增加该断点信息。

2.3.1.4 调试执行

启动调试

1. 设置完断点、运行配置后，单击工具栏中debug图标 即可启动调试功能。
2. 调试功能启用后，会停在第一个（已启用的）断点处，会自动在代码变量所在行右侧展示该变量的取值信息，调试界面如图2-18的绿色参数信息所示。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19行。
   - Variables栏展示变量信息以及线程信息。

图 2-18 调试工程界面



调试窗口图标解释如**表2-9**所示。

表 2-9 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**。
	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Frames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Frames信息和变量信息，详细操作请参见**Step Into功能**。

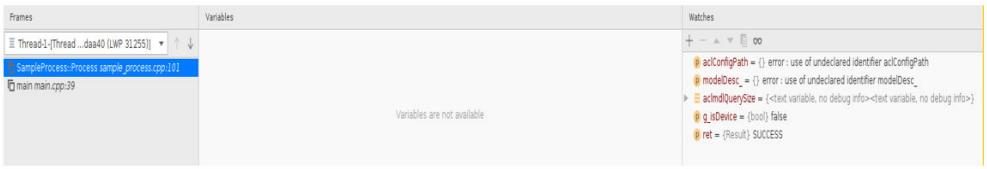
图标	名字	作用
	Force Step Into(Alt +Shift+F7)	强制进入函数。 单击按钮，调试时能进入任何函数，包括自定义函数以及官方库函数，
	Step Out(Shift+F8)	跳出函数。 如果调试时进入了某个函数，若检查该函数没有问题，则可以使用“Step Out”按钮跳出该函数，返回到原始位置。详细操作请参见**Step Out功能**。
	Run to cursor(Alt +F9)	运行到光标处。 可以在没有设置断点的情况下，让程序运行到光标所在行时暂停，同时会显示Frames信息和变量信息，详细操作请参见**Run to cursor功能**。

下面以具体示例介绍各个调试功能。

将变量添加到 Watches 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“Watches”区域中，如**图2-19**所示。

图 2-19 将变量添加到 Watches 区域

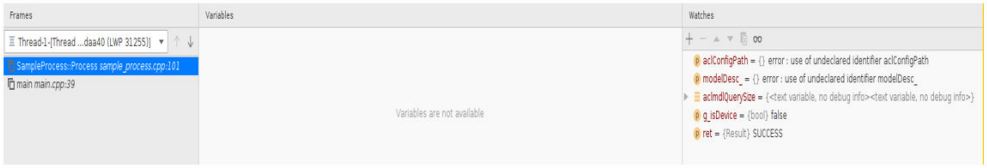


您也可以单击左侧工具栏的功能按钮，完成更多功能：

- 单击左侧工具栏  添加新的变量到“Watches”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Watches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2-20**所示。单击“Watches”窗口上方的  ，“Watches”窗口会自动合入“Variables”窗口。

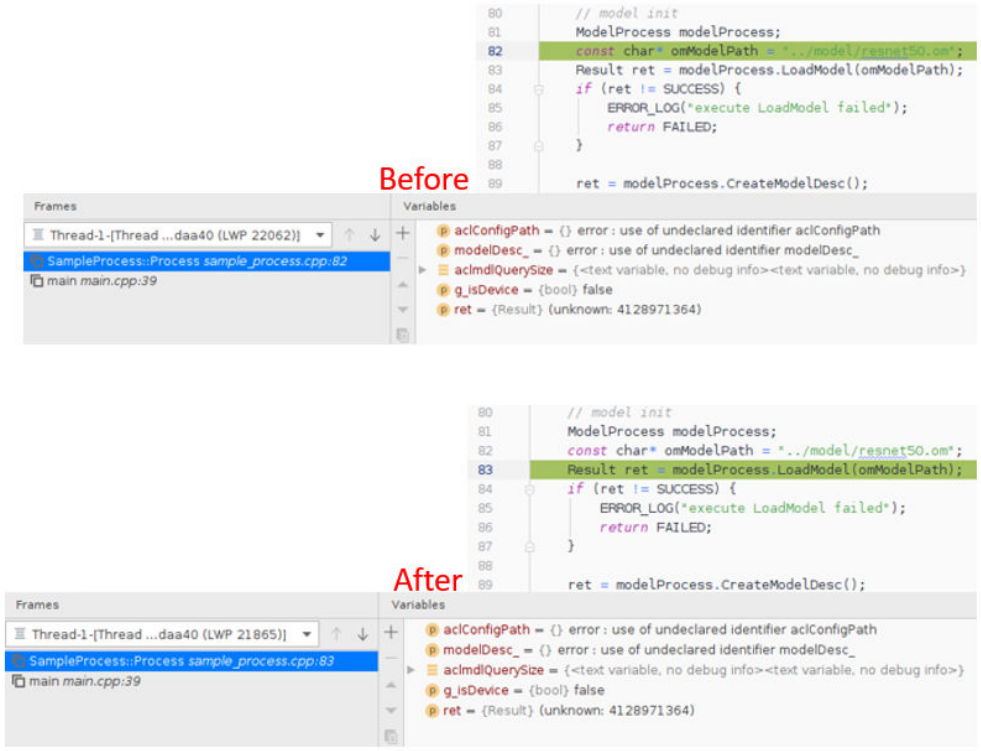
图 2-20 开启 Watches 窗口



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2-21所示。

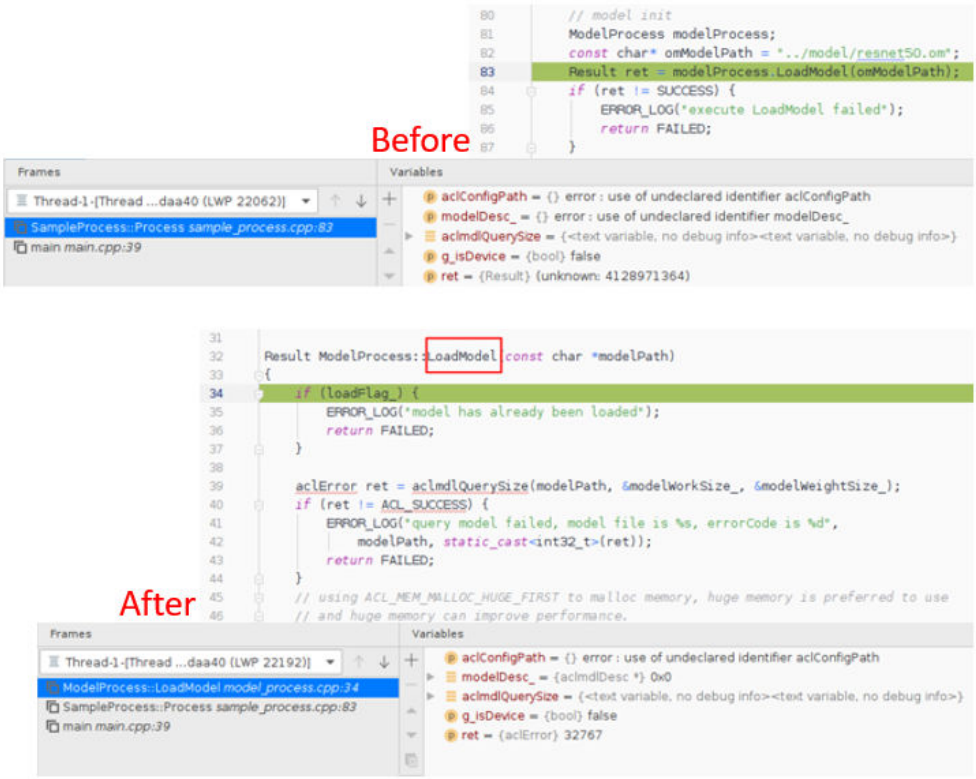
图 2-21 Step Over 功能



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2-22所示。

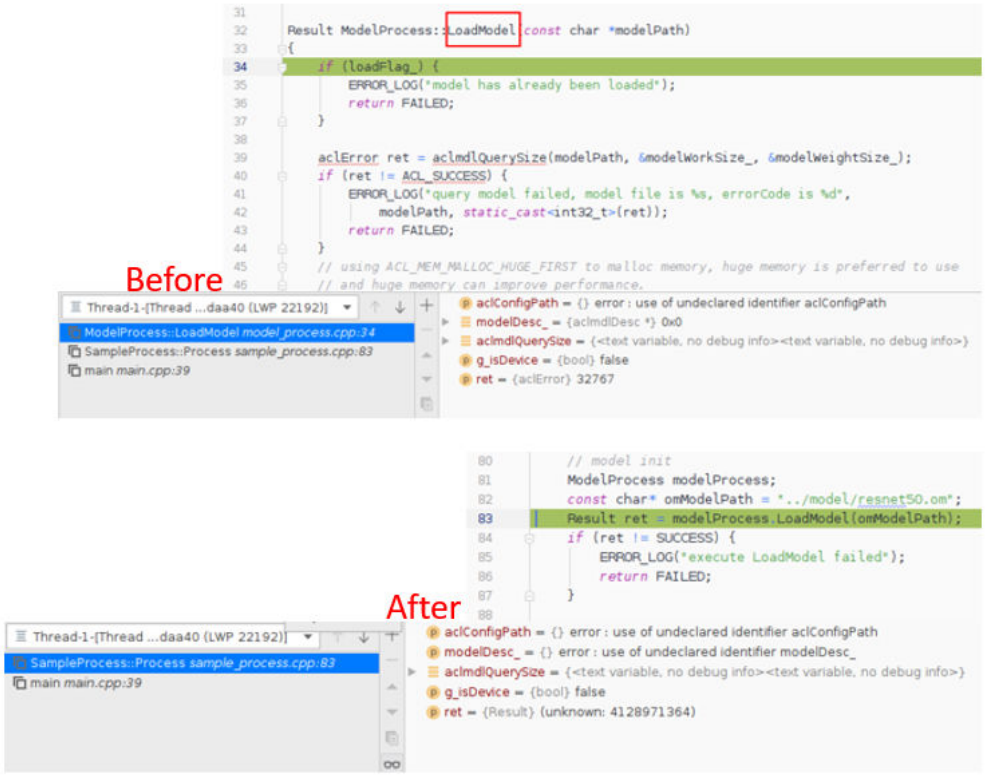
图 2-22 Step Into 功能



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2-23**所示。

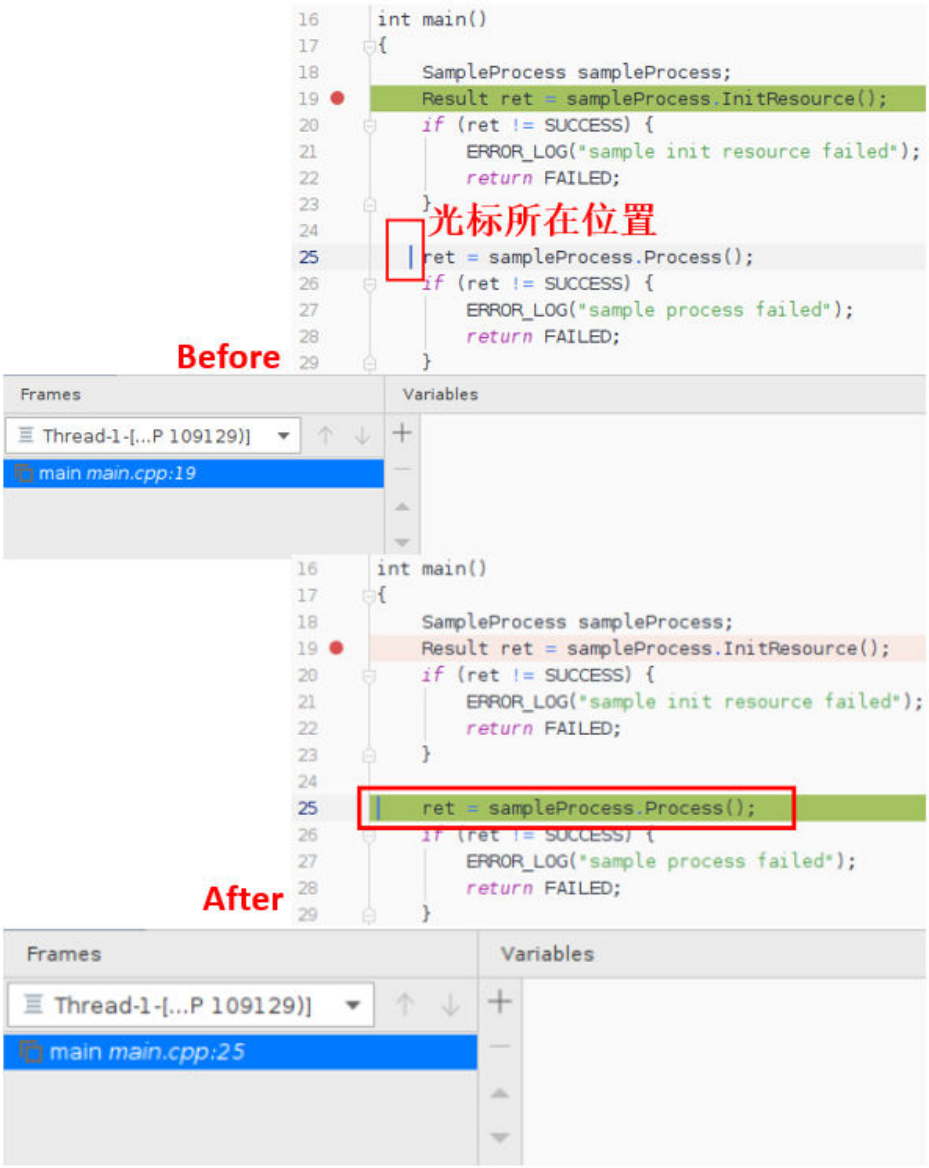
图 2-23 Step Out 功能



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2-24所示。

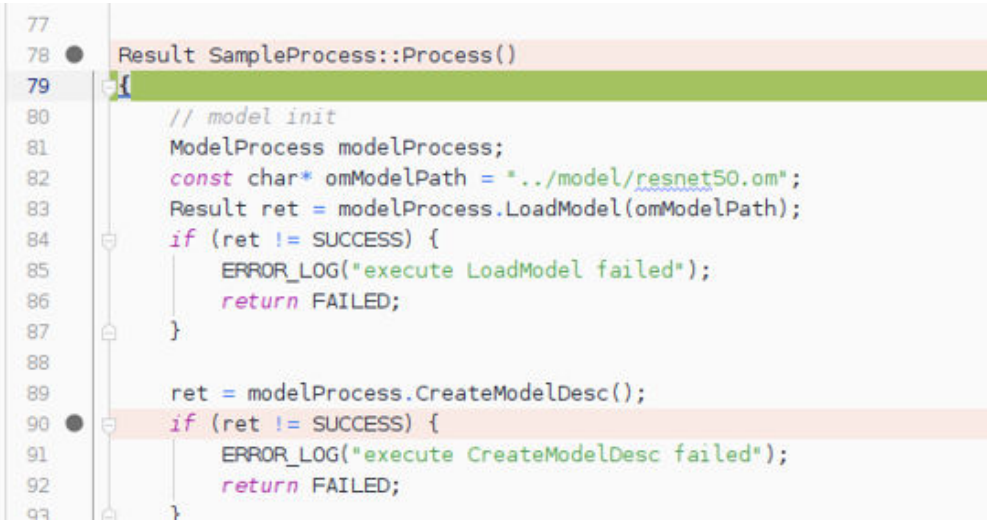
图 2-24 Run to cursor 功能



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2-25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 2-25 Mute Breakpoints 功能



2.3.2 调试 Python 代码

2.3.2.1 调试前准备

说明

Windows下支持远程Python代码的跳转、联想。

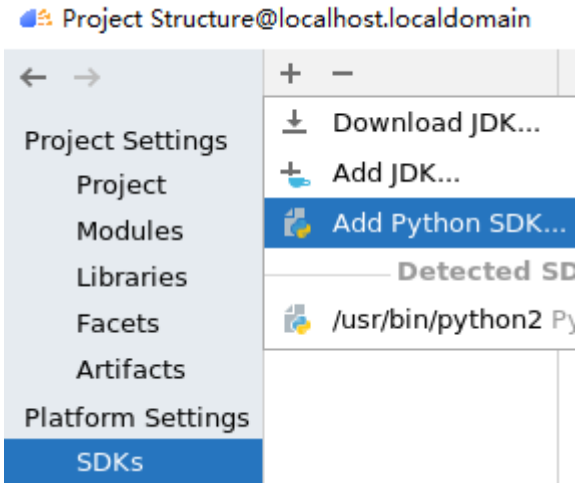
配置 Python SDK

为满足应用开发过程中Python代码调试的需求，提供基于Python Community Edition插件的Python代码调试方法，调试准备如下（以配置全局Python SDK为例，具体配置功能请参见**13.9 Python SDK设置**）：

步骤1 在顶部菜单栏中选择“File > Project Structure...”。

步骤2 在“Project Structure”左侧选择“Platform Settings > SDKs”，单击“+”，弹出框选择“Add Python SDK”，如**图2-26**所示。

图 2-26 配置 Python SDK



步骤3 在“Add Python Interpreter”窗口中，选择合适的Python解析器 ，用户自行添加。可选解析器如表2-10所示。单击“OK”完成添加。

图 2-27 Add Python Interpreter

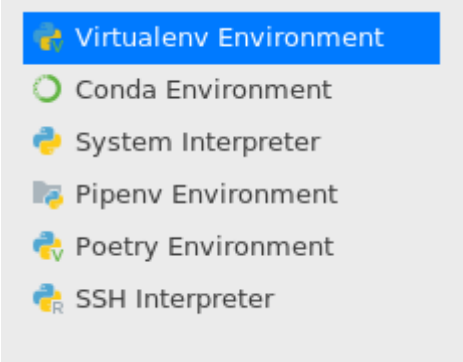


表 2-10 Python 解析器选项

解析器	说明
Virtualenv Environment	集成了Virtual Environment工具，用以创建独立的虚拟环境。
Conda Environment	使用Anaconda中带有的Python解释器。
System Interpreter	系统本地解析器。
Pipenv Environment	Pipenv是一种工具，提供了为Python项目创建虚拟环境所需的所有必要方法。
Poetry Environment	使用Poetry工具，创建基于工程依赖的虚拟环境。
SSH Interpreter	**远程解析器**。通过SSH方式添加远程设备的Python SDK并通过Deployment功能，将指定项目中的文件、文件夹同步到远程指定机器的指定目录，详细请参见13.3 Ascend Deployment。（如选择该选项，在工程运行时，MindStudio将以远端运行的方式进行。）

----结束

配置文件调试编译

步骤1 在顶部菜单栏中选择“Run > Edit Configurations...”或通过单击工具栏中的图2-28，进入运行配置界面。

图 2-28 运行配置框

步骤2 单击左上角的“+”，选择“Python”，新增文件的编译调试配置。

步骤3 右侧显示配置项，关键配置参数参见**表2-11**，配置示例如**图2-29**所示。

单击“Apply”应用后，单击“OK”保存并关闭调试配置。

图 2-29 源文件配置项

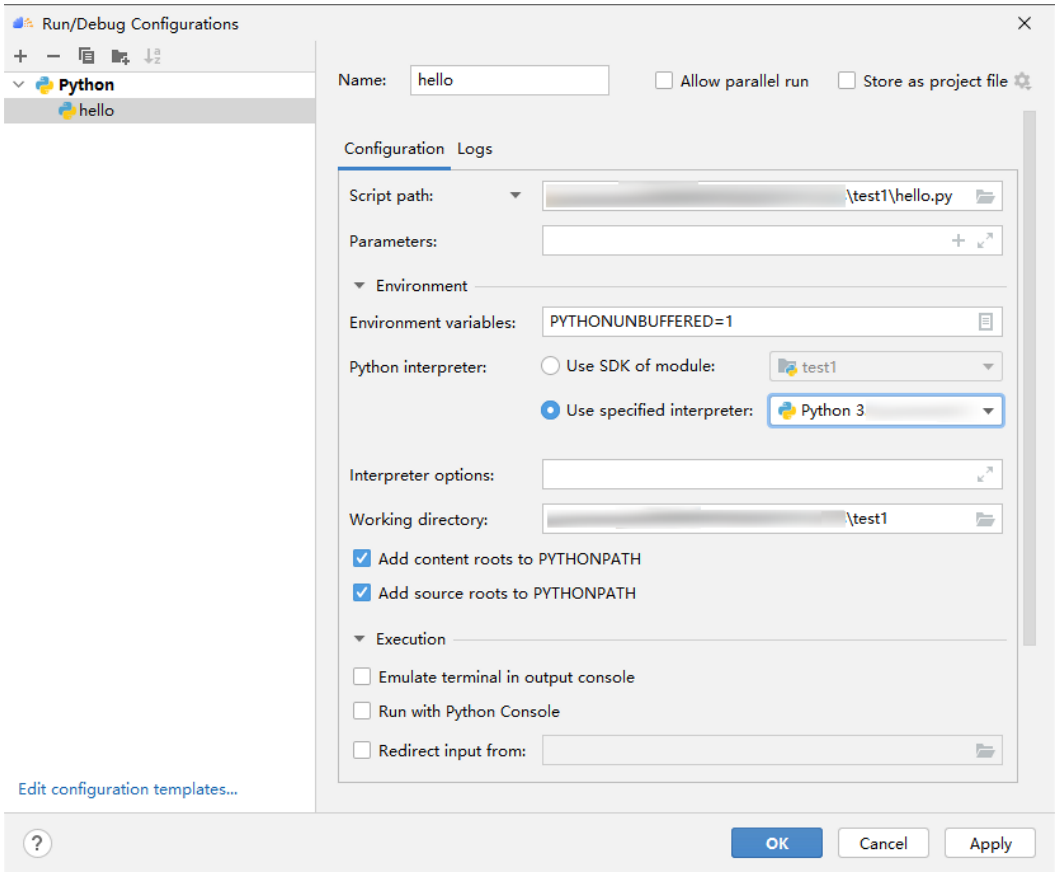


表 2-11 调试运行关键参数说明

参数	说明
Name	用户自行定义。
Script path	选择需要执行调试的Python源文件具体路径。
Python interpreter	“Use SDK of module”：使用模块级的Python SDK进行解析。 具体配置功能请参见**设置模块级Python SDK（昇腾工程）**或**设置模块级Python SDK（非昇腾工程）**。
	“Use specified interpreter”：使用已配置的特定解析器。 具体配置功能请参见**13.9 Python SDK设置**。
Working directory	工作目录（默认为需执行调试的Python源文件所在的目录）。

----结束

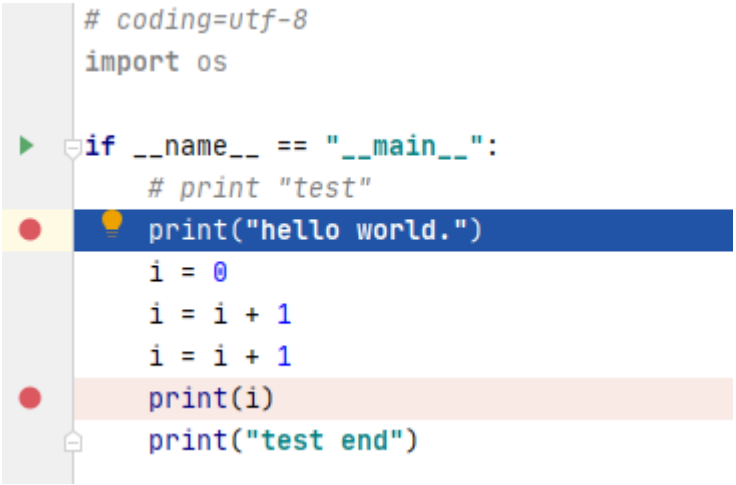
2.3.2.2 断点管理

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，然后用户可通过预先完成的设置，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2-30所示，断点添加成功后，左侧代码行号处会显示图标。

图 2-30 添加断点



删除断点

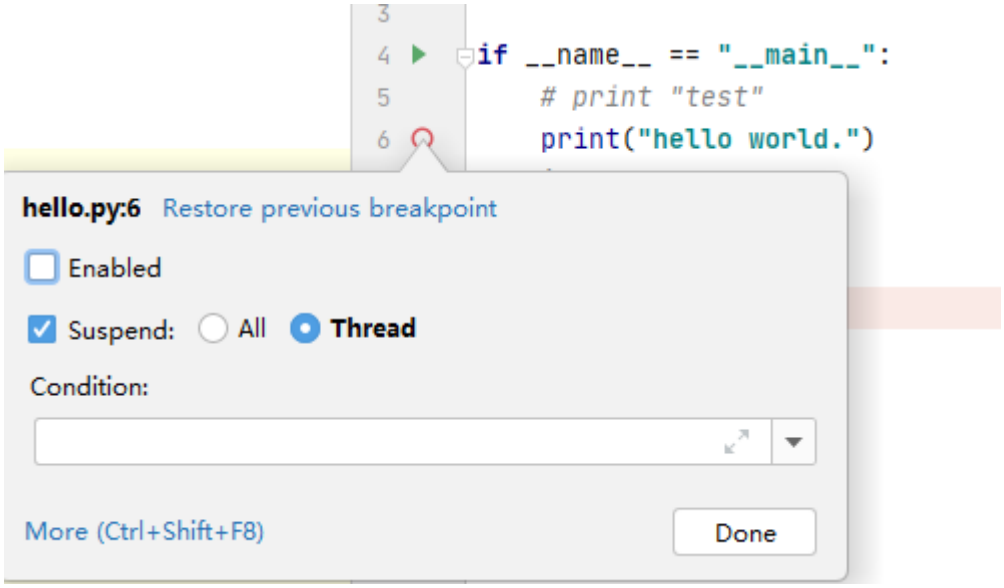
在设置断点的地方，单击左侧的即可取消断点。

停用断点

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，则可以通过以下操作暂时停用断点。

右击已经设置断点的代码行号处的，在弹出界面中不勾选“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 2-31 断点停用

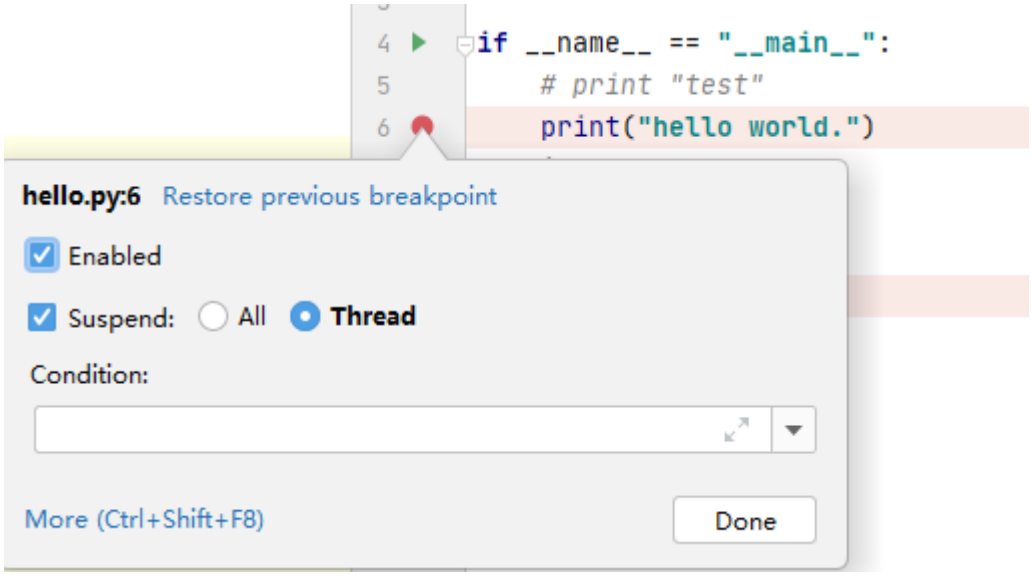


启用断点

若已经有部分断点设置了停用，执行新的调试功能时想重新启用断点，则可以通过以下操作实现。

右击已经停用断点处的，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 2-32 断点 Enable



说明

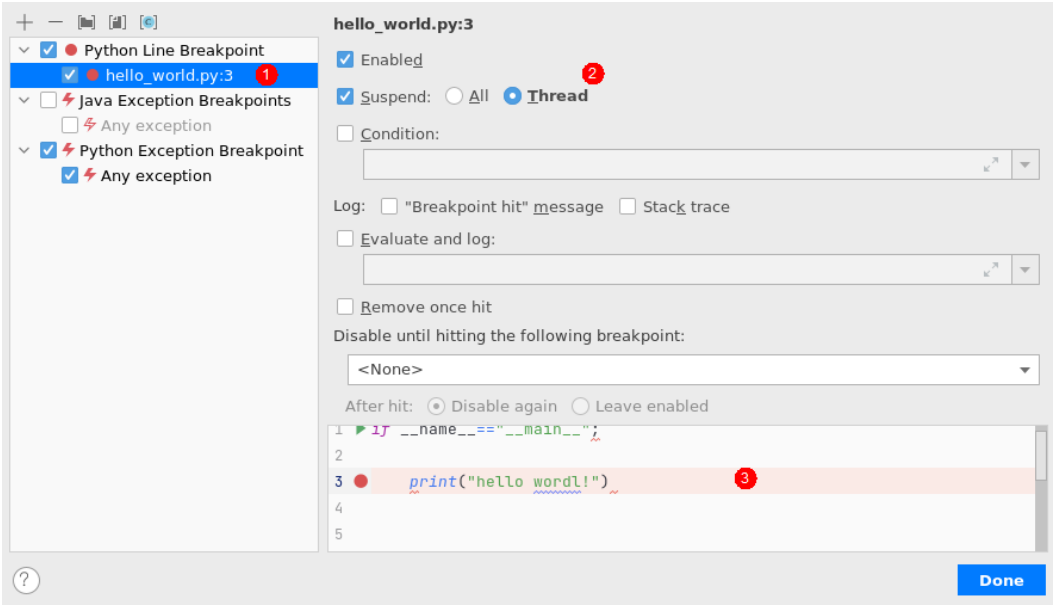
MindStudio重启以及工程重启后，不会影响当前工程中已经设置的断点状态和数量。

查看断点

在顶部菜单栏中找到“Run > View Breakpoints”，通过该功能，用户可以查看当前设置的所有断点。

单击断点界面的“More(Ctrl+Shift+F8)”，弹出图2-33所示断点查看界面。

图 2-33 断点查看界面



左侧序号1断点列表展示了当前代码中所设置的所有断点，序号2对应相应断点的停用（Disabled）或启用（Enabled）功能，序号3为代码预览区域。

功能使用方法参考如下：

- 单击左侧断点列表中，断点左侧的复选框，勾选表示该断点启用，相应序号2处“Enabled”功能被勾选，同时在代码预览区域中，对应代码行左侧行号处显示图标；取消勾选表示该断点停用，相应序号2处“Enable”功能被取消勾选，同时在代码预览区域中，对应代码行左侧行号处显示图标。
- 选中断点列表处的断点，单击上方的 − 可以删除断点，同时代码预览区域，相应代码行号处的断点图标随之删除。
- 在代码预览区域，代码行号处单击，则该行被标记为断点，同时断点列表中会自动增加该断点信息。

2.3.2.3 调试执行

启动调试

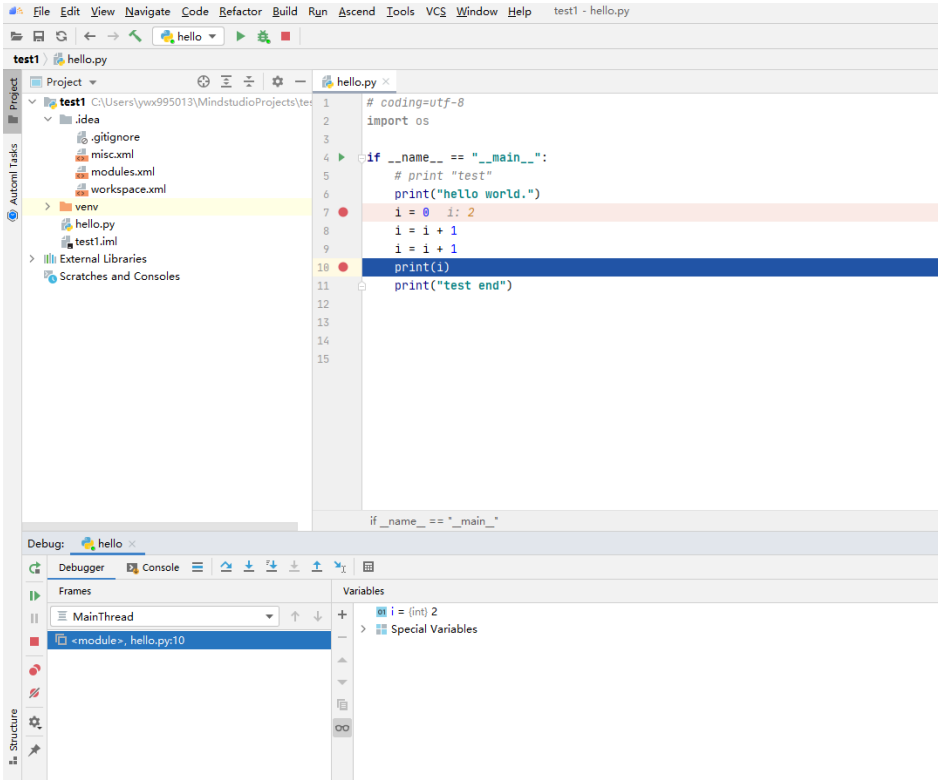
1. 设置完断点后，单击工具栏中debug图标即可启动调试功能。

   说明

   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录密码。

2. 调试功能启用后，会停在第一个（已启用的）断点处，调试界面如图2-34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。蓝色信息表示当前调试程序停留的代码行。
   - 右侧Variables展示变量信息以及线程信息。

图 2-34 调试工程界面

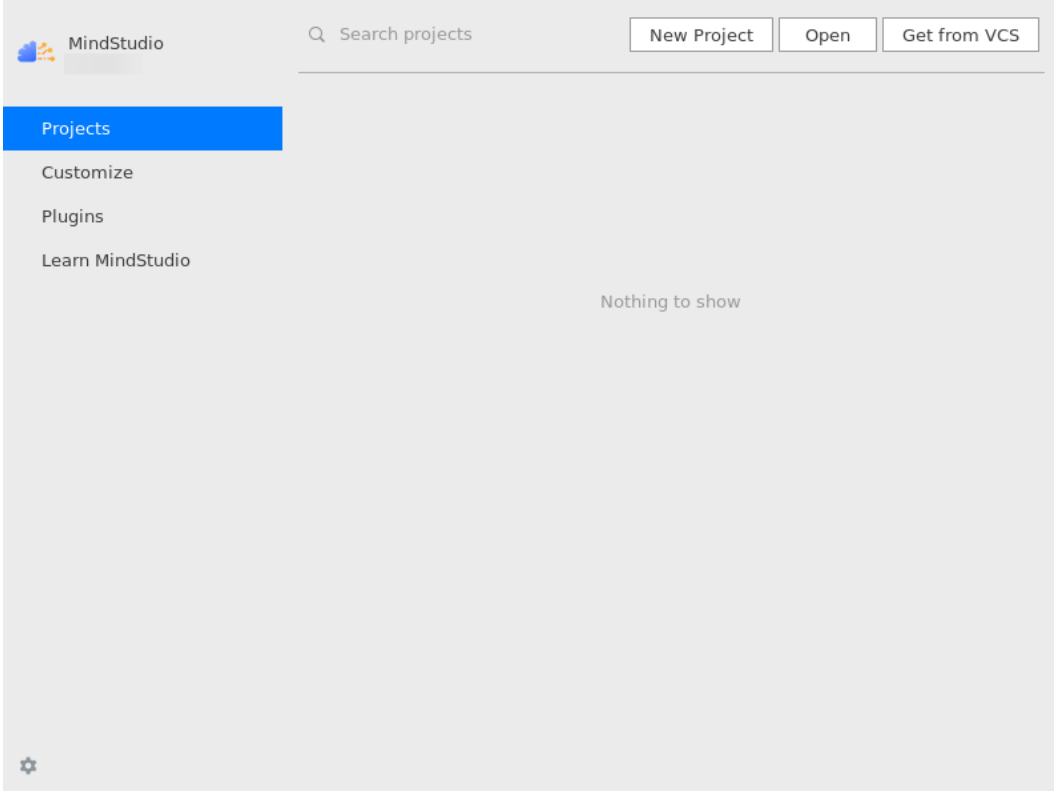


2.4 工程关闭

关闭工程

如需关闭当前打开的工程，可在顶部菜单栏中选择“File > Close Project”，关闭所有的工程后，系统会弹出如图2-35所示界面。您可以根据需要，根据右侧页面记录中打开最近的工程、单击“Open”导入现有工程或者单击“New Project”新建工程。

图 2-35 欢迎页面

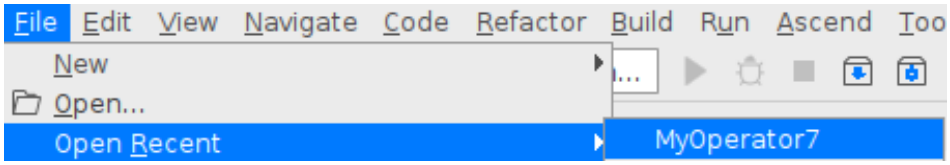


打开最近工程

工程关闭后，可通过如下任一方式，打开最近使用的工程：

- 依次单击顶部菜单栏中的“File > Open Recent > Manage Projects...”，在弹出页面中选择想要打开的工程。
- 依次单击顶部菜单栏中的“File > Open Recent”，在右侧弹出页面中选择想要打开的工程，如图2-36所示中的工程。

图 2-36 Open Project



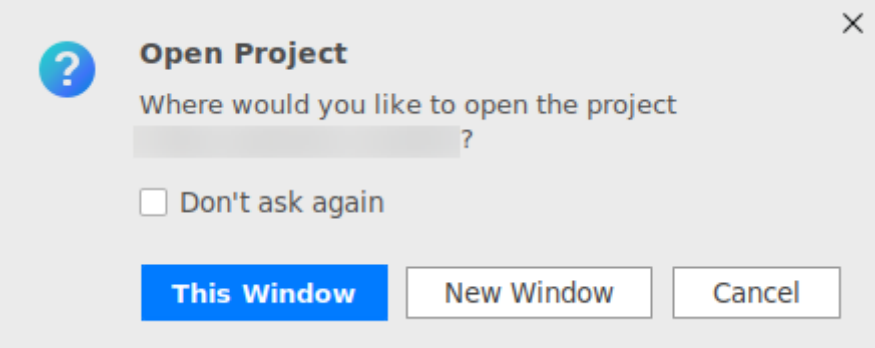
说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

若工作窗口已打开其他工程，会出现如图2-37所示提示。

图 2-37 工程创建提示



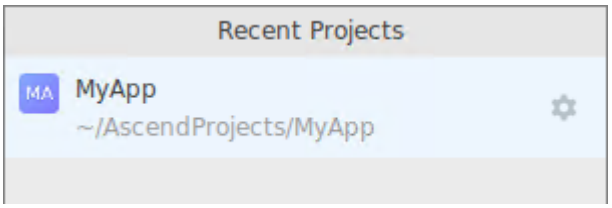
- 选择“This Window”，则直接在当前工作窗口打开新创建的工程。
- 选择“New Window”，则新建一个工作窗口打开新创建的工程。

移除最近工程记录

工程关闭后，可根据实际需求，选择移除最近打开的工程记录，具体操作可参考以下几种方式：

- 在MindStudio欢迎界面中，使鼠标移动至右侧想移除的工程上，单击工程项右侧的或直接右键单击选择“Remove from Recent Projects”。
- 在工程界面下，找到顶部菜单栏中的“File > Open Recent > Manage Projects...”，在弹出图2-38页面中使鼠标悬浮至想移除的工程上，单击右侧的或者右键单击选择“Remove from Recent Projects”。

图 2-38 Recent Projects

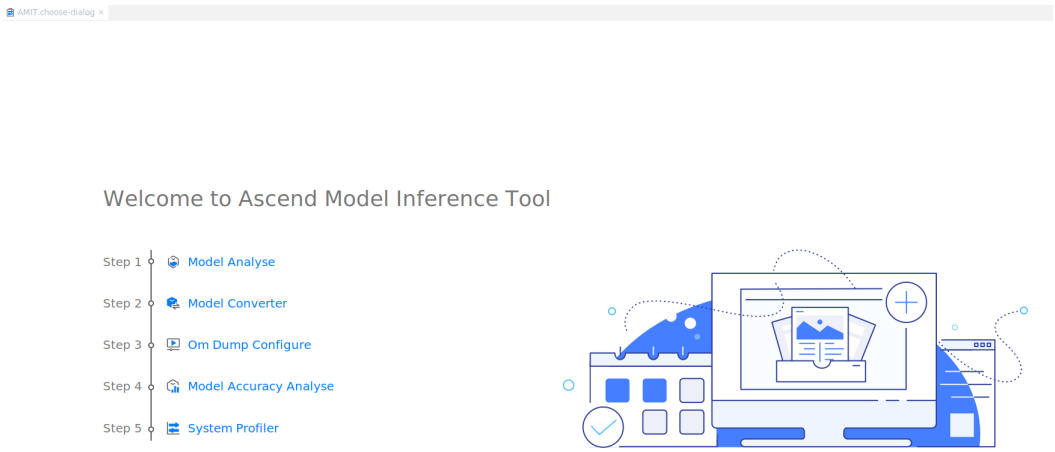


3 推理一体化工具

概述

为方便用户在模型推理过程中快速调用MindStudio功能，提供模型推理一体化入口AMIT（Ascend Model Inference Tool），支持用户在统一入口下调用模型分析、模型转换、离线模型dump、精度比对和性能分析工具，提升开发效率。同时，AMIT工具界面提供步骤指引，方便用户理解推理开发流程，更易上手使用。

图 3-1 AMIT 工具



1. 模型分析：分析模型中不支持的算子、查询当前CANN版本支持算子信息以及ModelZoo支持的模型信息。
2. 模型转换：提供ATC工具进行模型的转换，并生成OM离线模型。
3. 离线模型dump：根据转换后的OM离线模型生成dump数据。
4. 精度比对：根据OM离线模型dump数据与业界标准算子运算结果之间进行精度差异比对。
5. 性能分析：使用转换后的OM离线模型进行应用开发后，再使用Profiling工具采集性能数据并作简单的性能数据分析。

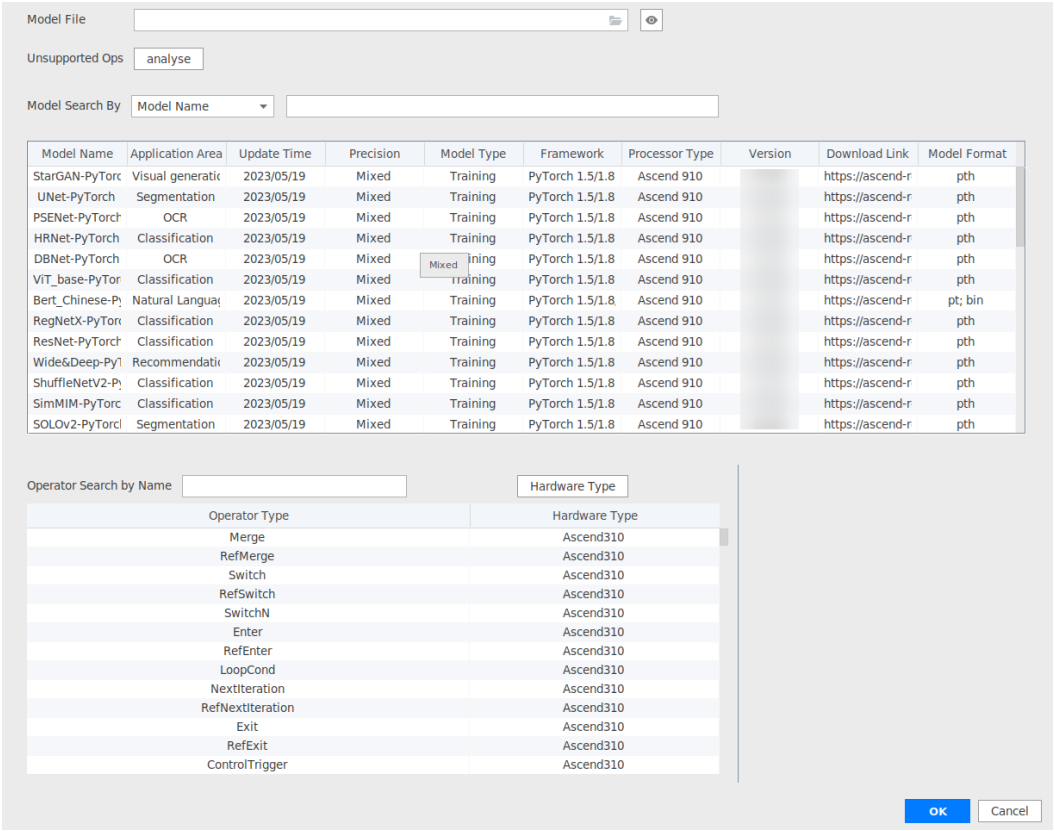
操作步骤

请参考以下步骤了解AMIT工具的操作流程，快速上手推理任务。

步骤1 在菜单栏选择“Ascend > AMIT”启动推理一体化工具。

步骤2 进入AMIT工具流程界面，单击“Model Analyse”启动模型分析任务，如图3-2所示。

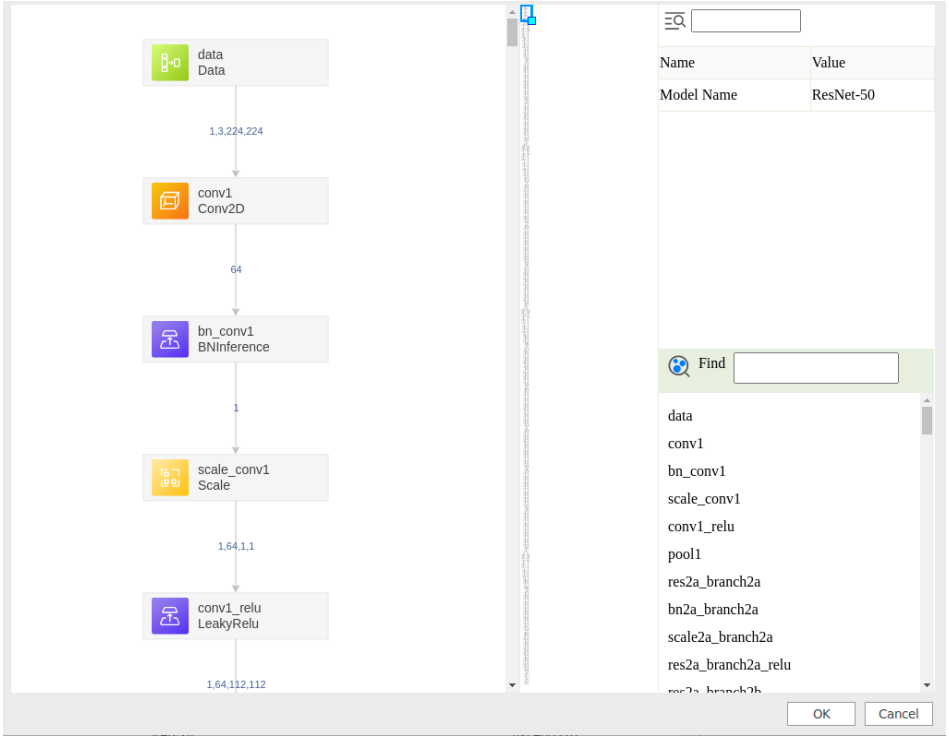
图 3-2 模型分析



1. 分析模型中不支持的算子信息。

   a. 在“Model File”中选择待分析模型，单击 查看可视化查看模型原始网络结构图，当前CANN版本不支持的算子会标红，如图3-3所示。更多操作请参考可视化界面说明。

图 3-3 模型可视化



b. 单击“analyse”，在“output”窗口输出CANN不支持的算子信息。若无内容输出，则表示支持模型内算子，可以进行转换和后续开发任务。

2. 模型速查，可以通过“Model Search by”后的下拉框选择支持的信息字段，并在输入框中补充关键字进行模型搜索，在当前界面得到查询结果，结果参数如表3-1所示。

图 3-4 模型速查

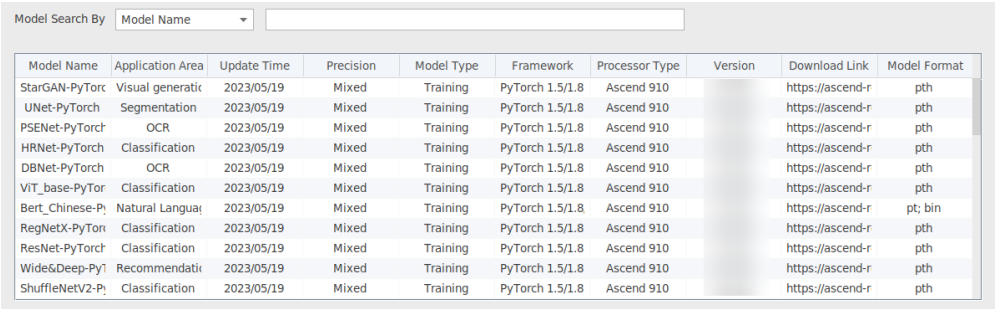


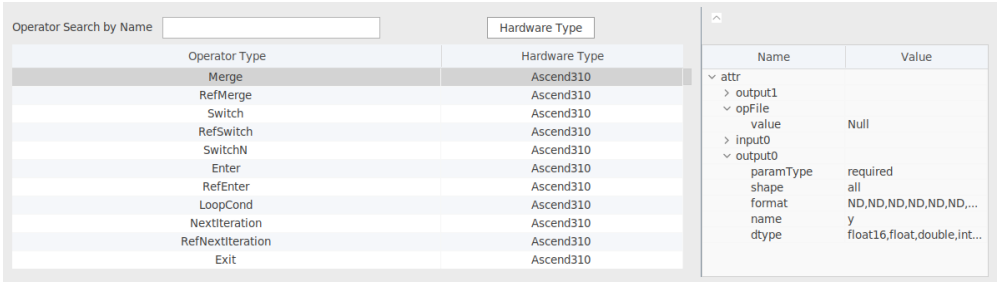
表 3-1 参数说明

参数	说明
Model Name	模型名称。
Application Area	应用领域，例如Image Classification、Object Detection。
Update Time	更新时间。
Precision	模型精度，例如FP16、INT8。

参数	说明
Model Type	模型类型，例如Offline Model。
Framework	框架，例如PyTorch、TensorFlow。
Processor Type	处理器类型。
Version	模型版本信息。
Download Link	下载链接，可以复制对应的链接粘贴到浏览器上打开，获取相应的模型文件。
Model Format	模型格式，例如om、onnx。

3. 算子速查，在“Operator Search by Name”输入需要查询的算子类型名称或硬件型号筛选需要查询的算子。选中单个算子类型时，在右侧查看算子具体信息。

图 3-5 算子具体信息



   - 算子类型（Operator Type）
   - 算子支持的硬件型号（Hardware Type）

步骤3 进入AMIT工具流程界面，单击“Model Converter”进入模型转换参数配置界面，工具使用方式请参考**模型转换**。转换后的OM离线模型可用于离线模型dump和性能分析任务。

步骤4 进入AMIT工具流程界面，单击“Om Dump Configure”进入离线模型dump选择模型界面，工具使用方式请参考**9.5.3.4 准备离线模型dump数据**。获取的离线模型dump数据可用于精度比对任务。

步骤5 进入AMIT工具流程界面，单击“Model Accuracy Analyse”进入精度比对参数配置界面，工具使用方式请参考**精度比对**。

步骤6 进入AMIT工具流程界面，单击“System Profiler”进入性能分析参数配置界面，工具使用方式请参考**性能分析**。

----结束

4 分析迁移

4.1 简介

分析迁移工具提供PyTorch/TensorFlow训练脚本一键式迁移至MindSpore或昇腾NPU的功能，开发者可做到少量代码修改或零代码完成迁移。详细信息请参见以下对应章节。

- **X2MindSpore**：将PyTorch/TensorFlow训练脚本迁移至可基于MindSpore运行的代码。
- **PyTorch GPU2Ascend**：分析PyTorch训练脚本的算子支持情况，支持将PyTorch训练脚本从GPU平台迁移至昇腾NPU平台。

说明

使用分析迁移工具迁移前，请用户自行确认原工程内各参数的正确性，需在原工程跑通的基础上使用工具进行迁移。

4.2 X2MindSpore

4.2.1 概述

X2MindSpore工具可将基于PyTorch和TensorFlow开发的模型及其训练脚本根据适配规则迁移为可基于MindSpore运行的代码，大幅度提高了脚本迁移速度，降低了开发者的工作量。

约束

- 该分析迁移工具支持对包括但不限于**4.2.4.1 模型列表**列出的模型进行迁移，迁移成功后可直接运行，部分模型需要根据实际情况进行少量适配。迁移后模型能够训练成功且收敛，但对最终精度和性能暂不做保证。

- X2MindSpore工具会在迁移后的目录中生成适配层文件目录x2ms_adapter。该目录中保存的是基于MindSpore实现的PyTorch/TensorFlow API替换接口，工具会将原PyTorch/TensorFlow API按照映射关系迁移至适配层API，迁移后代码依赖适配层运行。
- 通过该分析迁移工具迁移后的训练脚本支持在MindSpore 1.7-1.10版本上运行。
- MindSpore支持两种运行模式（Graph模式和PyNative模式），由于Graph模式存在Python语法限制，当前仅有表4-2中的ResNet、BiT系列（PyTorch）和UNet（PyTorch）的模型支持迁移至Graph模式，其余模型只支持迁移至PyNative模式，训练性能较Graph模式有所降低。具体差异详见MindSpore文档。
- 当前为了规避MindSpore中数据处理不支持创建Tensor的限制，在PyTorch迁移过程中将运行模式设置成了算子同步下发模式，可能存在训练性能的部分降低；用户可通过将context.set_context中的pynative_synchronize=True去除，使用算子异步下发模式提升性能；此时若报错，可检查数据处理部分代码，去除其中的创建Tensor行为，改为使用numpy的ndarray。
- 当其他程序占用当前卡，导致可用内存不足，可在训练脚本中context.set_context语句添加**device_id = ***，*为指定调用卡的ID。
- 当前TensorFlow 1.x的训练脚本不支持迁移为多卡脚本。

4.2.2 前提条件

迁移前须安装如下依赖。如下命令如果使用非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install pandas --user**。

```
pip3 install pandas       #pandas版本需大于或等于1.2.4
pip3 install libcst       #Python语法树解析器，用于解析Python文件
pip3 install jedi         #用于跨文件解析，建议安装
```

4.2.3 迁移操作

迁移步骤

步骤1 通过以下任一方式启动脚本迁移。

- 单击工具栏中  图标。
- 在菜单栏选择“Ascend > Migration Tools > X2MindSpore”。
- 右键单击工程目录中的文件夹，选择“X2MindSpore”。

步骤2 参数配置。

X2MindSpore启动后界面如图4-1所示，用户自行根据实际情况配置参数。

图 4-1 X2MindSpore 参数配置界面（开启 Distributed）

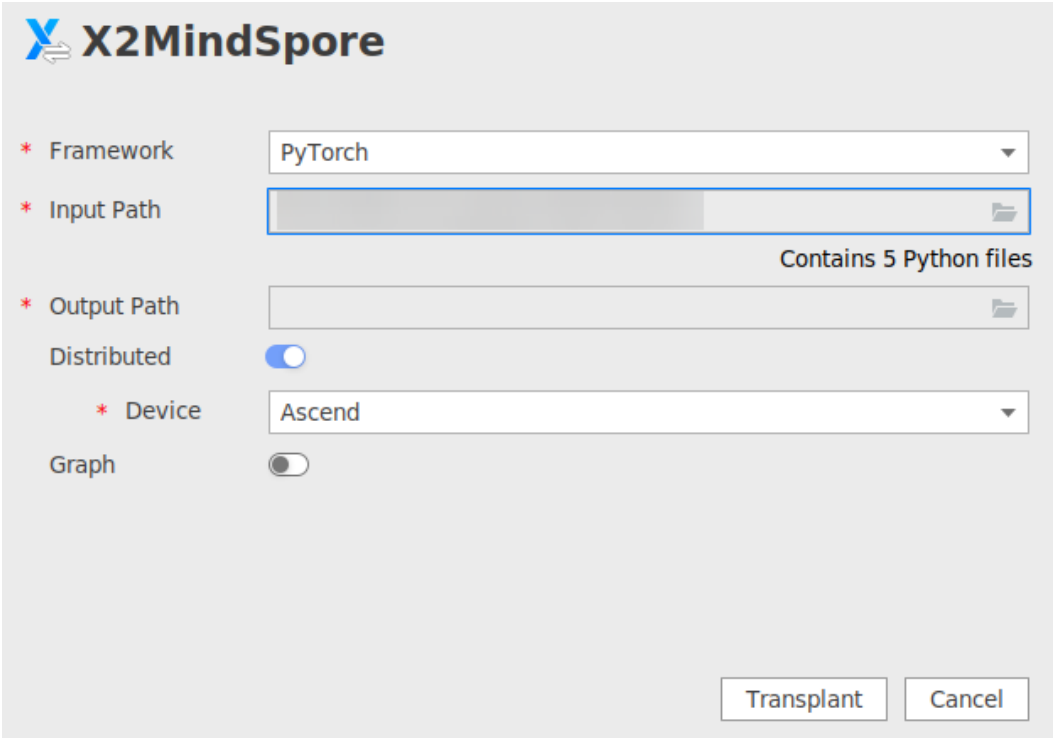


图 4-2 X2MindSpore 参数配置界面（开启 Graph）

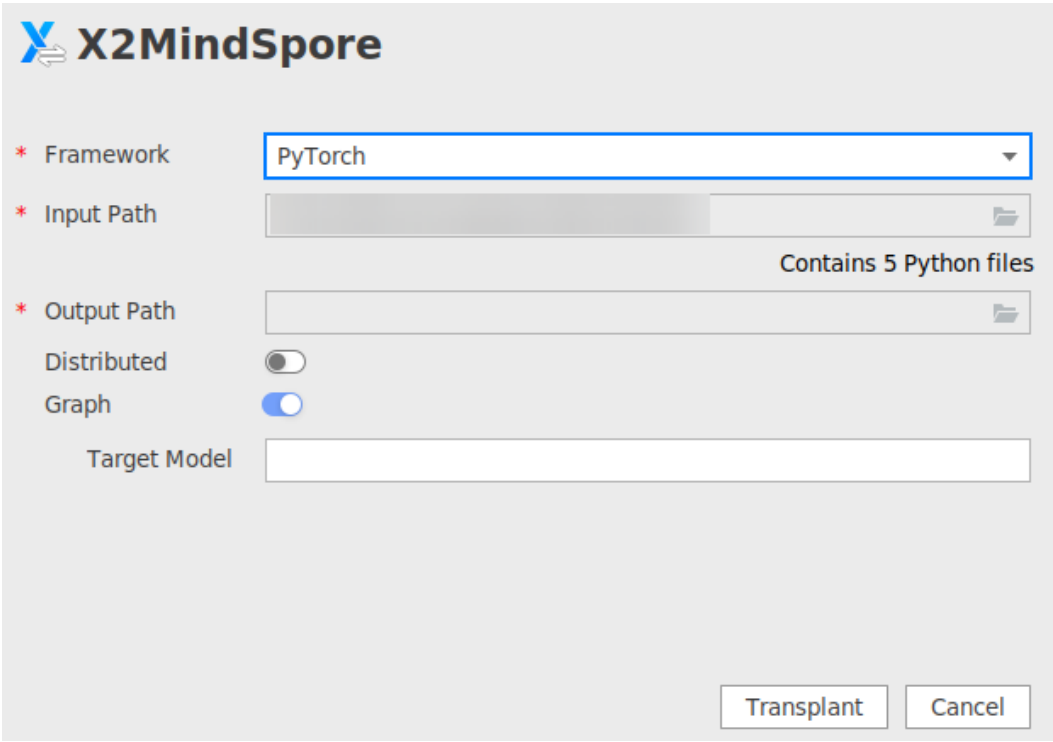


表 4-1 X2MindSpore 参数说明

参数	参数说明
Framework	要进行迁移的原始脚本的框架，必选。 取值如下。 • PyTorch（默认） • TensorFlow 1 • TensorFlow 2
Input Path	要进行迁移的原始工程目录。必选。
Output Path	脚本迁移结果文件输出路径。必选。 • 不开启“Distributed”即迁移至单卡脚本场景下，输出目录名为xxx_x2ms。 • 开启“Distributed”即迁移至多卡脚本场景下，输出目录名为xxx_x2ms_multi。 xxx为原始脚本所在文件夹名称。
Distributed	将GPU单卡脚本迁移为多卡脚本，仅支持“PyTorch”和“TensorFlow 2”框架。可选。默认关闭。 开启此参数后，需配置“Device”参数，将GPU单卡脚本迁移为指定设备的多卡脚本，取值如下： • Ascend（默认） • GPU
Graph	迁移后的脚本支持在MindSpore 1.8-1.10版本的Graph模式下运行。可选。默认关闭（即默认迁移至PyNative模式）。 目前该参数仅支持4.2.4.1 模型列表中ResNet、BiT系列和UNet的模型迁移至Graph模式，且不能与“Distributed”同时使用。 开启此参数后，可配置“Target Model”参数，即目标模型变量名，默认为“model”。

步骤3 单击“Transplant”，执行迁移任务。

完成后，**Output Path**输出目录下查看结果文件。

```
├── xxx_x2ms/xxx_x2ms_multi              // 脚本迁移结果输出目录
│   ├── 迁移后的脚本文件                  // 与迁移前的脚本文件目录结构一致。
│   ├── x2ms_adapter                    // 适配层文件。
│   ├── unsupported_api.csv             // 不支持API列表文件。
│   ├── custom_supported_api.csv        // 工具自定义适配API列表文件（目前仅支持PyTorch框架的训练脚本）。
│   ├── supported_api.csv               // 支持API列表文件。
│   ├── deleted_api.csv                 // 删除API列表文件。
│   ├── x2mindspore.log                 // 迁移日志，日志文件限制大小为1M，若超过限制将分多个文件进行存储，最多不会超过10个。
│   ├── run_distributed_ascend.sh       // 启用Distributed参数，且Device指定Ascend设备时，会生成该多卡启动shell脚本。
│   ├── rank_table_2pcs.json            // 启用Distributed参数，且Device指定Ascend设备时，会生成该2卡环境组网信息样例文件。
│   ├── rank_table_8pcs.json            // 启用Distributed参数，且Device指定Ascend设备时，会生成该8卡环境组网信息样例文件。
```

步骤4 在执行迁移后的模型文件前，请先将输出的工程路径加入环境变量PYTHONPATH中，示例如下。

```
export PYTHONPATH=${HOME}/output/xxx_x2ms:$PYTHONPATH
```

----结束

后续操作

- 如果启用了Distributed参数，迁移后需要根据Device指定的设备完成运行多卡脚本的特定操作。
  - Device指定Ascend设备。
    i. 参见**配置分布式环境变量**配置生成的多卡环境的组网信息json文件。
    ii. 将run_distributed_ascend.sh文件中的“please input your shell script here”语句替换为模型原训练脚本执行命令。

```
#!/bin/bash
echo
"============================================================================
========================================"
echo "Please run the script as: "
echo "bash run_distributed_ascend.sh RANK_TABLE_FILE RANK_SIZE RANK_START
DEVICE_START"
echo "For example: bash run_distributed_ascend.sh /path/rank_table.json 8 0 0"
echo "It is better to use the absolute path."
echo
"============================================================================
========================================"
execute_path=$(pwd)
echo "${execute_path}"
export RANK_TABLE_FILE=$1
export RANK_SIZE=$2
RANK_START=$3
DEVICE_START=$4
for((i=0;i<RANK_SIZE;i++));
do
  export RANK_ID=$((i+RANK_START))
  export DEVICE_ID=$((i+DEVICE_START))
  rm -rf "${execute_path}"/device_$RANK_ID
  mkdir "${execute_path}"/device_$RANK_ID
  cd "${execute_path}"/device_$RANK_ID || exit
  "please input your shell script here" > train$RANK_ID.log 2>&1 &
done
```

> **说明**
>
> 该脚本会在工程路径下创建device_*{RANK_ID}*目录，在该目录内去执行网络脚本，所以替换训练Python脚本执行命令时要注意训练Python脚本相对路径的变化。

    iii. 执行run_distributed_ascend.sh脚本以启动用户原工程。以8卡环境为例，命令如下。

```
bash run_distributed_ascend.sh RANK_TABLE_FILE RANK_SIZE RANK_START DEVICE_START
```

    - RANK_TABLE_FILE：多卡环境的组网信息json文件。
    - RANK_SIZE：指定调用卡的数量。
    - RANK_START：指定调用卡的逻辑起始ID，当前仅支持单机多卡，填0即可。
    - DEVICE_START：指定调用卡的物理起始ID。

  具体MindSpore分布式训练（Ascend）请参见**分布式并行训练（Ascend）**。

  - Device指定GPU设备。

    在GPU硬件平台上，MindSpore采用OpenMPI的mpirun进行分布式训练，可通过以下命令运行多卡脚本。

    ```
    mpirun -n {多卡脚本运行的GPU卡数量} {模型原来的训练shell脚本执行命令}
    ```

    具体MindSpore分布式训练（GPU）请参见**分布式并行训练（GPU）**。

- 如果启用了Graph参数，需要把训练脚本中WithLossCell类的construct函数修改为只包括模型的前向计算和loss计算，具体修改请参见迁移后脚本中的Transplant advice。
- 由于转换后的脚本与原始脚本框架不一致，迁移后的脚本在调试运行过程中可能会由于MindSpore框架的某些限制而抛出异常，导致进程终止，该类异常需要用户根据异常信息进一步调试解决。
- 分析迁移后可以参考**5 模型训练**进行训练。

4.2.4 附录

4.2.4.1 模型列表

注意：分析迁移工具的模型列表仅供参考，备注中示例的行数仅为参考，请以实际所在行数为准。

表 4-2 PyTorch 模型列表

序号	模型	原始训练工程代码链接参考	备注
1	3D-Transformer-tr_spe	**https://github.com/smiles724/Molformer/tree/f5cad25e037b0a63c7370c068a9c477f4004c5ea**	-
2	3D-Transformer-tr_cpe		
3	3D-Transformer-tr_full		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>4</td><td>AFM</td><td rowspan="16">https://github.com/shenweichen/DeepCTR-Torch/tree/b4d8181e86c2165722fa9331bc16185832596232</td><td rowspan="16">• 由于除DIN外，其它模型没有对应训练脚本，迁移前需要拷贝./examples/run_din.py文件，将其命名为run_<模型名称>.py，并做如下修改：
1. 导入模型结构，如from deepctr_torch.models.ccpm import CCPM。
2. 根据模型结构传入不同的入参初始化网络，如model = CCPM(feature_columns, feature_columns, device=device)。
3. 根据网络是否支持dense_feature，修改网络的输入。
• 在多卡训练时需要保持输入值与预测长度一致，迁移后若要进行多卡训练需进行如下配置：
1. 修改./deepctr_torch/models/basemodel.py中370行。
修改前：
eval_result[name] = metric_fun(y, pred_ans)
修改后：
eval_result[name] = metric_fun(y[:len(pred_ans)], pred_ans)
2. 修改./examples/run_classification_criteo.py中78，79行。
修改前：
print("test LogLoss", round(log_loss(test[target].values, pred_ans), 4))
print("test AUC", round(roc_auc_score(test[target].values, pred_ans), 4))
修改后：
print("test LogLoss", round(log_loss(test[target].values[:len(pred_ans)], pred_ans), 4))
print("test AUC", round(roc_auc_score(test[target].values[:len(pred_ans)], pred_ans), 4))</td></tr>
<tr><td>5</td><td>AutoInt</td></tr>
<tr><td>6</td><td>CCPM</td></tr>
<tr><td>7</td><td>DCN</td></tr>
<tr><td>8</td><td>DeepFM</td></tr>
<tr><td>9</td><td>DIN</td></tr>
<tr><td>10</td><td>FiBiNET</td></tr>
<tr><td>11</td><td>MLR</td></tr>
<tr><td>12</td><td>NFM</td></tr>
<tr><td>13</td><td>ONN</td></tr>
<tr><td>14</td><td>PNN</td></tr>
<tr><td>15</td><td>WDL</td></tr>
<tr><td>16</td><td>xDeepFM</td></tr>
</table>

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>17</td><td>AFN</td><td rowspan="4">https://github.com/shenweichen/DeepCTR-Torch/tree/2cd84f305cb50e0fd235c0f0dd5605c81148840a2</td><td rowspan="4">• 因为模型没有对应训练脚本，迁移前需要将example/run_classification_criteo.py中DeepFM网络改为需测试的网络。
修改run_classification_criteo.py文件54行。
修改前：
model = DeepFM(linear_feature_columns=linear_feature_columns, dnn_feature_columns=dnn_feature_columns,
根据模型结构传入不同的入参初始化网络，如AFN模型修改后：
model = AFN(linear_feature_columns=linear_feature_columns, dnn_feature_columns=dnn_feature_columns,
• 在多卡训练时需要保持输入值与预测长度一致，迁移后若要进行多卡训练需进行如下配置：
1. 修改./deepctr_torch/models/basemodel.py中370行。
修改前：
eval_result[name] = metric_fun(y, pred_ans)
修改后：
eval_result[name] = metric_fun(y[:len(pred_ans)], pred_ans)
2. 修改./examples/run_classification_criteo.py中78，79行。
修改前：
print("test LogLoss", round(log_loss(test[target].values, pred_ans), 4))
print("test AUC", round(roc_auc_score(test[target].values, pred_ans), 4))
修改后：
print("test LogLoss", round(log_loss(test[target].values[:len(pred_ans)], pred_ans), 4))
print("test AUC", round(roc_auc_score(test[target].values[:len(pred_ans)], pred_ans), 4))</td></tr>
<tr><td>18</td><td>DCNMix</td></tr>
<tr><td>19</td><td>DIFM</td></tr>
<tr><td>20</td><td>IFM</td></tr>
<tr><td>21</td><td>BERT</td><td>https://github.com/codertimo/BERT-pytorch/tree/d10dc4f9d5a6f2ca74380f62039526eb7277c671</td><td>• 迁移完成后，该工程在需要安装才能使用，安装步骤如下：
– 去除requirements.txt文件中的torch项。
– 执行python3 setup.py install。
• 具体使用方式详见仓库README。</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注
22	BEiT	https://github.com/microsoft/unilm/tree/9cbfb3e40eedad33a8d2f1f15c4a1e26fa50a5b1	• 迁移前进行以下操作。 – 把模型源码下载后只保留beit文件夹。 – 下载开源代码库pytorch-image-models0.3.3版本的代码，将其中的timm文件夹移至beit文件夹下 • 迁移后，由于不能将PyTorch模型权重迁移为MindSpore模型权重，需要注释utils.py的第550和560行代码。
23	BiT-M-R101x1	https://github.com/google-research/big_transfer/tree/140de6e704fd8d61f3e5ea20ffde130b7d5fd065	• 数据集如果使用cifar-10-bin，可从https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz获取。 • 数据集如果使用cifar-100-bin，可从https://www.cs.toronto.edu/~kriz/cifar-100-binary.tar.gz获取。
24	BiT-M-R101x3		
25	BiT-M-R152x2		
26	BiT-M-R152x4		
27	BiT-M-R50x1		
28	BiT-M-R50x3		
29	BiT-S-R101x1		
30	BiT-S-R101x3		
31	BiT-S-R152x2		
32	BiT-S-R152x4		
33	BiT-S-R50x1		
34	BiT-S-R50x3		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>35</td><td>ADMMSLIM</td><td rowspan="56">https://github.com/RUCAIBox/RecBole/tree/6e66565347a71c6f9662f9e7366a55c35be4fa46</td><td rowspan="56">

在迁移前，BERT4REC、CASER、HGN、REPEATNET模型需要参考官网新建run.py，添加模型代码，然后单独迁移run.py文件。
在迁移完成后需要修改以下内容。

1. 注释./recbole/utils/utils.py中的第267行。
<pre># model = copy.deepcopy(model)</pre>
2. 修改./recbole/data/dataloader/general_dataloader.py中的第254行。
修改前：
<pre>uid_list = list(user_df[self.uid_field])</pre>
修改后：
<pre>uid_list = list(user_df[self.uid_field].asnumpy())</pre>
3. 修改./recbole/data/interaction.py中的第140行。
修改前：
<pre>ret[k] = self.interaction[k][index]</pre>
修改后：
<pre>if self.interaction[k][index].ndim == 0:
    ret[k] = self.interaction[k][index].reshape(1)
else:
    ret[k] = self.interaction[k][index]</pre>
4. 在./recbole/quick_start/quick_start.py中第41行后增加如下内容。
<pre>import mindspore.context as context
from x2ms_adapter.core.context import x2ms_context
if not x2ms_context.is_context_init:

context.set_context(mode=context.PYNATIVE_MODE, pynative_synchronize=True)
    x2ms_context.is_context_init = True</pre>
5. 仅DGCF模型需要执行该步骤。为满足显存要求，在./recbole /properties/model/DGCF.yaml中配置n_factors为1。
6. 仅FISM、DSSM、FFM和FWFM模型需要执行该步骤。
修改./recbole/model/general_recommender/fism.py中的75行和./recbole/model/context_aware_recommender/路径下dssm.py中的60行、ffm.py中的78行、fwfm.py中的64行。
修改前：
<pre>self.bceloss = loss_wrapper.BCEWithLogitsLoss()</pre>
修改后：
<pre>self.bceloss = loss_wrapper.BCELoss()</pre>
</td></tr>
<tr><td>36</td><td>BERT4REC</td></tr>
<tr><td>37</td><td>BPR</td></tr>
<tr><td>38</td><td>CASER</td></tr>
<tr><td>39</td><td>CDAE</td></tr>
<tr><td>40</td><td>CFKG</td></tr>
<tr><td>41</td><td>CKE</td></tr>
<tr><td>42</td><td>CONVNCF</td></tr>
<tr><td>43</td><td>DGCF</td></tr>
<tr><td>44</td><td>DMF</td></tr>
<tr><td>45</td><td>DSSM</td></tr>
<tr><td>46</td><td>EASE</td></tr>
<tr><td>47</td><td>ENMF</td></tr>
<tr><td>48</td><td>FFM</td></tr>
<tr><td>49</td><td>FISM</td></tr>
<tr><td>50</td><td>FWFM</td></tr>
<tr><td>51</td><td>GCMC</td></tr>
<tr><td>52</td><td>HGN</td></tr>
<tr><td>53</td><td>KGCN</td></tr>
<tr><td>54</td><td>NCL</td></tr>
<tr><td>55</td><td>REPEATNET</td></tr>
<tr><td>56</td><td>SLIMELASTIC</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注
57	CenterNet-ResNet50	https://github.com/bubbliiiing/centernet-pytorch/tree/91b63b9d0fef2e249fbddee8266c79377f0c7946	• 迁移后根据仓库readme处理数据集。 • 由于没有训练好的mindspore模型权重，因此需要将train.py中的model_path置为空。
58	CenterNet-Hourglass Net		
59	Conformer-tiny	https://github.com/pengzhiliang/Conformer/tree/815aaad3ef5dbdfcf1e11368891416c2d7478cb1	• 迁移前需要将timm库（推荐0.3.2版本）放到原始代码根目录下。 • 由于框架限制，当前不支持--repeated-aug，所以训练时需要使用--no-repeated-aug参数。
60	Conformer-small		
61	Conformer-base		
62	DeiT-tiny		
63	DeiT-small		
64	DeiT-base		
65	CvT-13	https://github.com/microsoft/CvT/tree/f851e681966390779b71380d2600b52360ff4fe1	• 迁移前需要将timm库（推荐0.3.2版本）和einops库放到原始代码根目录下。 • 迁移前修改./run.sh中内容: – 将train()中训练启动方式（4~10行）改为**python3 tools/train.py ${EXTRA_ARGS}**。 – 将test()中测试启动方式（15~21行）改为**python3 tools/test.py ${EXTRA_ARGS}**。
66	CvT-21		
67	CvT-W24		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>68</td><td>albert-base-v1</td><td rowspan="13">https://github.com/huggingface/transformers/tree/49cd736a288a315d741e5c337790effa4c9fa689</td><td rowspan="15">

迁移前，需要把原仓库的模板文件移走，这些文件本质不是python文件却以.py后缀命名。
mv templates ../
迁移后，请进行以下修改：

为避免出现list out of range错误，对src/transformers/configuration_utils.py的d["torch_dtype"] = x2ms_adapter.tensor_api.split(str(d["torch_dtype"]), ".")[1]语句取消索引的使用：
修改后：
d["torch_dtype"] = x2ms_adapter.tensor_api.split(str(d["torch_dtype"]), ".")
对src/transformers/modeling_utils.py的model_to_save.config.torch_dtype = x2ms_adapter.tensor_api.split(str(dtype), ".")[1]语句取消索引的使用：
修改后：
model_to_save.config.torch_dtype = x2ms_adapter.tensor_api.split(str(dtype), ".")
将./src/transformers/utils/import_utils.py中的is_torch_available()定义返回值改为“True”来走原来的PyTorch流程：
修改前：
def is_torch_available():
  return _torch_available
修改后：
def is_torch_available():
  return True

</td></tr>
<tr><td>69</td><td>albert-large-v1</td></tr>
<tr><td>70</td><td>albert-xlarge-v1</td></tr>
<tr><td>71</td><td>albert-xxlarge-v1</td></tr>
<tr><td>72</td><td>albert-Text classification</td></tr>
<tr><td>73</td><td>albert-TokenClassification</td></tr>
<tr><td>74</td><td>albert-QA</td></tr>
<tr><td>75</td><td>albert-MultipleChoice</td></tr>
<tr><td>76</td><td>bert-base-uncased</td></tr>
<tr><td>77</td><td>bert-large-uncased</td></tr>
<tr><td>78</td><td>bert-base-QA</td></tr>
<tr><td>79</td><td>bert-base-Text classification</td></tr>
<tr><td>80</td><td>bert-base-MultipleChoice</td></tr>
<tr><td>81</td><td>bert-base-token-classification</td><td rowspan="2">https://github.com/huggingface/transformers/tree/49cd736a288a315d741e5c337790effa4c9fa689</td></tr>
<tr><td>82</td><td>distilbert-base-uncased</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注
83	distilbert-base-QA		
84	distilbert-base-Text classification		
85	roberta-base		
86	roberta-large		
87	roberta-base-Multiple Choice		
88	roberta-base-Text classification		
89	roberta-base-token-classification		
90	roberta-base-QA	https://github.com/huggingface/transformers/tree/49cd736a288a315d741e5c337790effa4c9fa689	
91	xlm-mlm-en-2048		
92	xlm-mlm-ende-1024		
93	xlm-mlm-enro-1024		
94	xlm-clm-enfr-1024		
95	xlm-Text classification		
96	xlm-Roberta-base		

序号	模型	原始训练工程代码链接参考	备注
97	xlm-roberta-large		
98	xlm-roberta-Text classification		
99	Xlm-reberta-token-classification		
100	xlm-roberta-QA		
101	xlnet-base-cased		
102	xlnet-large-cased		
103	XLNet-base-Text classification		
104	XLNet-base-token-classification		
105	XLNet-base-Multiple Choice		
106	XLNet-base-QA		

序号	模型	原始训练工程代码链接参考	备注

107	DistilRoBERTa	https://github.com/huggingface/transformers/tree/49cd736a288a315d741e5c337790effa4c9fa689	迁移后，请进行以下修改：
			修改./src/transformers/utils/import_utils.py中的is_torch_available()定义。
			修改前：
			`def is_torch_available():`
			`  return _torch_available`
			修改后：
			`def is_torch_available():`
			`  return True`

108	Transform-XL	https://github.com/huggingface/transformers/tree/49cd736a288a315d741e5c337790effa4c9fa689	迁移后，请进行以下修改：
			• 修改./src/transformers/utils/import_utils.py中的is_torch_available()定义。
			修改前：
			`def is_torch_available():`
			`  return _torch_available`
			修改后：
			`def is_torch_available():`
			`  return True`
			• MindSpore的dtype转换为字符串类型后的结构和torch的有所不同，因此需要对./src/transformers/modeling_utils.py进行以下修改。
			修改前：
			`model_to_save.config.torch_dtype = x2ms_adapter.tensor_api.split(str(dtype), ".")[1]`
			修改后：
			`model_to_save.config.torch_dtype = str(dtype)`

109	ConvLSTM	https://github.com/jhhuang96/ConvLSTM-PyTorch/tree/d44942983c05c66381eeb9f54e88c828e9e37cfc	• 使用数据集./data/train-images-idx3-ubyte.gz进行训练。
			• 迁移后需要将./ConvRNN.py文件中所有sigmoid函数替换为tanh。示例如下：
			修改前：
			`z = x2ms_adapter.sigmoid(zgate)`
			修改后：
			`z = x2ms_adapter.tanh(zgate)`

序号	模型	原始训练工程代码链接参考	备注

110	DeepCTR-Deep & Cross Network	https://github.com/shenweichen/DeepCTR-Torch/tree/2cd84f305cb50e0fd235c0f0dd5605c8114840a2	• 获取Criteo数据集，对数据集进行解压处理后，用于模型训练。
			• 迁移后需要修改./examples/run_classification_criteo.py文件：
			1. 新增导入如下内容：
			`from x2ms_adapter.torch_api.optimizers import optim_register`
			2. 修改数据集路径，示例如下。
			`data = pd.read_csv('数据集路径')`
			3. 将模型名称修改为待训练模型，参考如下加粗字体信息修改。
			`model = DCN(linear_feature_columns=linear_feature_columns, dnn_feature_columns=dnn_feature_columns, task='binary', l2_reg_embedding=1e-5, device=device)`
			`model.compile(optim_register.adagrad(x2ms_adapter.parameters(model),lr=1e-5, weight_decay=0.0001), "binary_crossentropy",metrics=["binary_crossentropy", "auc"], )`
			`history = model.fit(train_model_input, train[target].values, batch_size=512, epochs=10, verbose=2,validation_split=0.2)`
			`pred_ans = model.predict(test_model_input, 512)`

111	DeepCTR-Deep & Wide	https://github.com/shenweichen/DeepCTR-Torch/tree/2cd84f305cb50e0fd235c0f0dd5605c8114840a2	• 获取MovieLens数据集，对数据集进行解压处理后，用于模型训练。
			• 迁移后修改./examples/run_multivalue_movielens.py文件。
			1. 修改导入内容。
			修改前：
			`from tensorflow.python.keras.preprocessing.sequence import pad_sequences`
			修改后：
			`from tensorflow.keras.preprocessing.sequence import pad_sequences`
			2. 新增导入如下内容：
			`from deepctr_torch.models import WDL`
			3. 修改数据集路径，示例如下。
			`data = pd.read_csv('数据集路径')`
			4. 修改为待训练模型，参考如下加粗字体信息修改。
			`model = WDL(linear_feature_columns, dnn_feature_columns, task='regression', device=device)`
			`model.compile("adam", "mse", metrics=['mse'], )`
			`history = model.fit(model_input, data[target].values, batch_size=256, epochs=10, verbose=2, validation_split=0.2)`

序号	模型	原始训练工程代码链接参考	备注
112	EfficientNet-B0	https://github.com/lukemelas/EfficientNet-PyTorch/tree/7e8b0d312162f335785fb5dcfa1df29a75a1783a	-
113	EfficientNet-B1		
114	EfficientNet-B2		
115	EfficientNet-B3		
116	EfficientNet-B4		
117	EfficientNet-B5		
118	EfficientNet-B6		
119	EfficientNet-B7		
120	EfficientNet-B8		
121	egfr-att	https://github.com/lehgtrung/egfr-att/tree/0666ee90532b1b1a7a2a179f8fbf10af1fdf862f	-

序号	模型	原始训练工程代码链接参考	备注

122	FasterRCNN	https://github.com/AlphaJia/pytorch-faster-rcnn/tree/943ef668facaacf77a4822fe79331343a6ebca2d	• 支持以下backbone网络:
			– mobilenet
			– resnet-fpn
			– vgg16
			– HRNet
			• 迁移前，进行如下修改。
			– 因为用到torchvision 0.9.0的MultiScaleRoIAlign算子，因此要将该算子所在文件torchvision/ops/poolers.py拷贝到根目录下，且将./utils/train_utils.py和./utils/faster_rcnn_utils.py中用到该算子的地方修改为如下内容。
			`from poolers import MultiScaleRoIAlign`
			– 由于MindSpore没有torch.utils.data.Subset对应的API，需将./utils/coco_utils.py中涉及该API的代码注释掉，示例如下。
			`# if isinstance(dataset, torch.utils.data.Subset):`
			`#  dataset = dataset.dataset`
			• 迁移后，修改如下内容。
			由于MindSpore中的BitwiseOr算子不支持UINT8的输入，需对./utils/roi_header_util.py的如下表达式进行修改。
			修改前:
			`pos_inds_img | neg_inds_img`
			修改后:
			`pos_inds_img.astype(mindspore.int32) | neg_inds_img.astype(mindspore.int32)`

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>123</td><td>FCOS-ResNet50</td><td rowspan="2">https://github.com/zhenghao977/FCOS-PyTorch-37.2AP/tree/2bfa4b6ca57358f52f7bc7b44f506608e99894e6</td><td rowspan="2">迁移后需要进行以下修改。
• 数据集使用VOC数据集，需要修改./train_voc.py代码中第39行数据集路径为实际路径。
• 由于mindspore中没有对应的scatter算子，因此需要对./model/loss.py文件进行以下修改。
1. 将125和126行替换为以下代码：
min_indices = mindspore.ops.ArgMinWithValue(-1)(areas.reshape(-1, areas.shape[-1]))
tmp = np.arange(0, batch_size * h_mul_w).astype(np.int32)
indices = mindspore.ops.Concat(-1)((mindspore.ops.ExpandDims()(mindspore.Tensor(tmp), -1), mindspore.ops.ExpandDims()(min_indices[0], -1)))
reg_targets = mindspore.ops.GatherNd()(ltrb_off.reshape(-1, m, 4), indices)
2. 将130行替换为以下代码：
cls_targets = mindspore.ops.GatherNd()(classes.reshape(-1, m, 1), indices)
3. 在文件的第7行导入相应的包：
import numpy as np
• 由于没有mindspore的预训练模型，因此需要将./model/config.py中的pretrained，freeze_stage_1 ，freeze_bn修改为False。</td></tr>
<tr><td>124</td><td>FCOS-ResNet101</td></tr>
<tr><td>125</td><td>MGN-strong</td><td>https://git.openi.org.cn/Smart_City_Model_Zoo/mgn-strong</td><td>1. 该模型依赖于torchvision，因此需要将torchvision/目录下的models/文件夹拷贝至./mgn-strong/model/目录下；
2. 将./mgn-strong/model/models/__init__.py的内容改为：
from .resnet import *
3. 修改./mgn-strong/model/mgn.py第7行的import语句：
修改前：
from torchvision.models.resnet import resnet50, Bottleneck, resnet101
修改后：
from .models.resnet import resnet50, Bottleneck, resnet101
4. 修改./mgn-strong/loss/triplet.py第83行addmm_调用语句：
修改前：
dist.addmm_(1, -2, inputs, inputs.t())
修改后：
dist.addmm_(inputs, inputs.t(), beta=1, alpha=-2)
5. 需要在Mindspore 1.7.0上运行。</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注

126	MobileNet V1 SSD	https://github.com/qfgaohao/pytorch-ssd/tree/f61ab424d09bf3d4bb3925693579ac0a92541b0d	MindSpore暂不支持数据集加载中使用Tensor和在模型中对ModuleList使用切片。因此迁移前需要对原始工程文件夹下./vision/ssd/ssd.py进行如下修改。
			• 第57行for循环修改为如下循环。
			`for idx in range(start_layer_index, end_layer_index):`
			`  layer = self.base_net[idx]`
			• 第143行“**self.center_form_priors = center_form_priors**”语句前插入“**center_form_priors = center_form_priors.asnumpy()**”。

127	MobileNet V1 SSD-Lite		
128	MobileNet V2 SSD-Lite		
129	MobileNet V3-Large SSD-Lite		
130	MobileNet V3-Small SSD-Lite		
131	SqueezeNet SSD-Lite		
132	VGG16 SSD		
133	SqueezeNet	https://github.com/weiaicunzai/pytorch-cifar100/tree/2149cb57f517c6e5fa7262f958652227225d125b	• 数据集使用cifar-100-bin，可从https://www.cs.toronto.edu/~kriz/cifar-100-binary.tar.gz获取。 • 根据实际修改./utils.py文件中的数据集路径。
134	InceptionV3		
135	InceptionV4		
136	InceptionResNetV2		
137	Xception		
138	Attention56		
139	Stochastic Depth18		
140	Stochastic Depth34		
141	Stochastic Depth50		

序号	模型	原始训练工程代码链接参考	备注
142	StochasticDepth101		
143	VGG11		
144	VGG13		
145	VGG16		
146	DenseNet161		
147	DenseNet169		
148	DenseNet201		
149	PreActResNet34		
150	PreActResNet50		
151	PreActResNet101		
152	PreActResNet152		
153	ResNeXt152		
154	SEResNet34		
155	SEResNet50		
156	SEResNet101		
157	VGG19	https://github.com/kuangliu/pytorch-cifar/tree/49b7aa97b0c12fe0d4054e670403a16b6b834ddd	• 数据集需要使用cifar-10-bin，可从https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz获取。 • 切换不同的网络需要在./main.py中修改使用的网络名称。
158	PreActResNet18		
159	DenseNet121		

序号	模型	原始训练工程代码链接参考	备注
160	ResNeXt29_2x64d		
161	MobileNet		
162	MobileNetV2		
163	SENet18		
164	ShuffleNetG2		
165	GoogleNet		
166	DPN92		

167	RetineNet-ResNet34	https://github.com/yhenon/pytorch-retinanet/tree/0348a9d57b279e3b5b235461b472d37da5feec3d	• 由于原始仓库代码中有关于torch版本和加载torch模型的代码，迁移前需要修改原始工程脚本./train.py：
			– 第77~88行的backbone模型选择，**pretrained**参数都设置为**False**。
			– 删除第18行**assert torch.__version__.split('.')[0] == '1'**
			• 迁移完成后，由于mindspore反向传递和数据集加载的限制，需要进行以下修改：在./retinanet/losses.py中25和26行替换成以下代码。
			`def print_grad_fn(cell_id, grad_input, grad_output):`
			`  pass`
			`class FocalLoss(mindspore.nn.Cell):`
			`  def __init__(self):`
			`  super(FocalLoss, self).__init__()`
			`  self.register_backward_hook(print_grad_fn)`

168	RetineNet-ResNet50		
169	Res2Net	https://github.com/Res2Net/Res2Net-ImageNet-Training/tree/d77c16ff111522c64e918900f100699acc62f706	暂不支持torchvision.models相关接口的迁移，需做以下操作。 修改原始工程： 1. 创建目录./res2net_pami/models。 2. 在./res2net_pami/main.py中，将**import torchvision.models as models**改为**import models**。

序号	模型	原始训练工程代码链接参考	备注
170	ResNet18	https://github.com/pytorch/examples/tree/41b035f2f8faede544174cfd82960b7b407723eb/imagenet	暂不支持torchvision.models相关接口的迁移，需做以下操作。 修改原始工程： 1. 创建目录./imagenet/models。 2. 从torchvision库（0.6.0版本）中拷贝torchvision/models/resnet.py至./imagenet/models下，删除**from .utils import load_state_dict_from_url**语句。 3. 创建./imagenet/models/__init__.py文件，内容为： `from .resnet import *` 4. ./main.py中，将**import torchvision.models as models**改为**import models**。
171	ResNet34		
172	ResNet50		
173	ResNet101		
174	ResNet152		
175	ResNeXt-50（32x4d）		
176	ResNeXt-101（32x8d）		
177	Wide ResNet-50-2		
178	Wide ResNet-101-2		

序号	模型	原始训练工程代码链接参考	备注

179	sparse_rcnnv1-resnet50	https://github.com/lianghеming/sparse_rcnnv1/tree/65f54808f43c34639085b01f7ebc839a3335a386	迁移后，修改如下内容。
			• ./x2ms_adapter.torch_api.nn_api.nn.py中手动修改MultiheadAttention类中初始化函数的batch_size、src_seq_length和tgt_seq_length的值。
			• 对./nets/common.py的**if x.requires_grad:**语句改为**if True:**。
			• 对./losses/sparse_rcnn_loss.py中linear_sum_assignment函数的item[i]变量转换成numpy。
			`indices = linear_sum_assignment(item[i].asnumpy())`
			• 对./datasets/coco.py做2点修改：
			– __getitem__定义模块的返回语句**return box_info**修改为**return box_info.img,box_info.labels,box_info.boxes**。
			– 修改collect_fn定义模块的for循环。
			`image,labels,boxes = item #增加`
			`img = (image[:, :, ::-1] / 255.0 - np.array(rgb_mean)) / np.array(rgb_std)#item.img改为image`
			`target = x2ms_np.concatenate([labels[:, None], boxes], axis=-1)#item.labels改为labels，item.boxes改为boxes`

180	sparse_rcnnv1-resnet101		
181	ShuffleNetV2	https://github.com/megvii-model/ShuffleNet-Series/tree/aa91feb71b01f28d0b8da3533d20a3edb11b1810	-
182	ShuffleNetV2+		

183	SMSD	https://git.openi.org.cn/PCLNLP/Sarcasm_Detection/src/commit/54bae1f2306a4d730551b4508ef502cfdbe79918	迁移前需要进行以下操作：
			• 在./SMSD/目录中新建state_dict文件夹。
			• 在./SMSD/models/__init__.py中添加如下语句导入SMSD_bi模型。
			`from models.SMSD_bi import SMSD_bi`
			运行迁移后代码可通过--repeat参数控制训练重复次数（以SMSD_bi模型为例）：
			`python3 train.py --model_name SMSD_bi --repeat 1`

184	SMSD_bi		

序号	模型	原始训练工程代码链接参考	备注
185	Swin-Transformer	https://github.com/microsoft/Swin-Transformer/tree/5d2aede42b4b12cb0e7a2448b58820aeda604426	• 迁移前需要将timm库代码放到原始代码根目录下。 • timm库版本推荐0.4.12。 • 当前--cfg参数只支持以下四个配置文件： – swin_tiny_patch4_window7_224.yaml – swin_tiny_c24_patch4_window8_256.yaml – swin_small_patch4_window7_224.yaml – swin_base_patch4_window7_224.yaml
186	Transformer	https://github.com/SamLynnEvans/Transformer/tree/e06ae2810f119c75aa345854428720268675e6462	需要对于该代码仓中脚本依赖的torchtext库进行迁移并有如下注意事项： • 拷贝迁移后的torchtext_x2ms到脚本文件夹。 • 将torchtext_x2ms重命名为torchtext，以保证用户调用的是迁移后的torchtext。 • torchtext版本建议使用0.6.0。

187	UNet	https://github.com/milesial/Pytorch-UNet/tree/e1a69e7c6ce18edd47271b01e4aabc03b436753d	进行静态图模式迁移前，需要进行以下操作：
			1. 修改./utils/dice_score.py第14行。
			修改前：
			`if sets_sum.item() == 0:`
			修改后：
			`if sets_sum == 0:`
			2. 修改./train.py中第95行F.one_hot(true_masks, net.n_classes).permute(0, 3, 1, 2).float(), 中**net.n_classes**修改为171行中**n_classes**赋的值。

序号	模型	原始训练工程代码链接参考	备注
188	RCNN-Unet	https://github.com/bigmb/Unet-Segmentation-Pytorch-Nest-of-Unets/tree/c050f5eab6778cba6dcd8f8a68b74c9e62a698c8	迁移前需要进行以下操作： • 由于MindSpore求导存在语法限制，./pytorch_run.py中249和252行的注释需要修改为4空格倍数对齐。 • 模型要求输入图片大小为16的倍数，因此当数据集图片大小不满足16倍数时，需取消./pytorch_run.py中121、122行和505、506行的注释，将图片缩放裁剪为16倍数。 • 当数据集label图片通道为1时，需要在./pytorch_run.py的293行尾加入.convert('RGB')将图片转换为3通道。 • 由于MindSpore中使用ModuleList会导致子层的权重名称改变，需要将./pytorch_run.py第350行的torch.nn.ModuleList改为list，避免checkpoint文件保存后无法重新加载。
189	Attention Unet		
190	RCNN-Attention Unet		
191	Nested Unet		
192	ViT-B_16	https://github.com/jeonsworld/ViT-pytorch/tree/460a162767de1722a014ed2261463dbbc01196b6	数据集需要使用cifar-10-bin，可从https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz获取。
193	ViT-B_32		
194	ViT-L_16		
195	ViT-L_32		
196	ViT-H_14		
197	R50-ViT-B_16		

序号	模型	原始训练工程代码链接参考	备注

198	YOLOR-CSP	https://github.com/WongKinYiu/yolor/tree/462858e8737f56388f812cfe381a69c4ffca0cc7	迁移完成后需进行如下修改：
			1. 修改./utils/datasets.py文件。
			修改前：
			`if os.path.isfile(cache_path):`
			修改后：
			`if False:`
			2. 修改./models/models.py中的create_module函数。
			修改前：
			`module_list[j][0].bias = mindspore.Parameter(bias_, ⋯)`
			修改后：
			`module_list[j][0].bias = mindspore.Parameter(bias.reshape(bias_.shape),⋯)`
			3. 修改./utils/loss.py中的build_targets函数。
			修改前：
			`gij = x2ms_adapter.tensor_api.long((⋯))`
			`gi, gj = gij.T`
			`⋯`
			`tbox.append(⋯)`
			修改后：
			`gij = x2ms_adapter.tensor_api.long((⋯)).T`
			`gi, gj = gij`
			`⋯`
			`gij = gij.T`
			`tbox.append(⋯)`
			4. 修改./train.py文件。
			修改前：
			`if '.bias' in k:`
			修改后：
			`if '.bias' in k or '.beta' in k:`

199	YOLOR-CSP-X		
200	YOLOR-P6		
201	YOLOR-W6		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>202</td><td>YOLOv3</td><td rowspan="3">https://github.com/ultralytics/yolov3/tree/ae37b2daa74c599d640a7b9698eeafd64265f999</td><td rowspan="3">迁移完成后进行如下修改。
• 对./models/yolo.py：
class Detect(mindspore.nn.Cell):
stride = None #删除
onnx_dynamic = False
def __init__(self, …):
…
self.stride = None #新增
• 对./utils/loss.py的build_targets函数：
修改前
gij = x2ms_adapter.tensor_api.long((…))
gi, gj = gij.T
…
tbox.append(…)
修改后
gij = x2ms_adapter.tensor_api.long((…)).T
gi, gj = gij
…
gij = gij.T
tbox.append(…)
• 对./val.py中run函数：
– 删除“path, shape = Path(paths[si]), shapes[si][0]”。
– 删除scale_coords函数的调用处。
– 删除“callbacks.run('on_val_image_end', …)”。
• 将./models/目录下对应模型配置文件{model_name}.yaml中所有nn.*修改为x2ms_adapter.torch_api.nn_api.nn.*。
• 运行多卡场景下，将train.py中“val_loader = create_dataloader(…）”的rect参数修改为False。</td></tr>
<tr><td>203</td><td>YOLOv3-Tiny</td></tr>
<tr><td>204</td><td>YOLOv3-SSP</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注

205	YOLOv4	https://github.com/WongKinYiu/PyTorch_YOLOv4/tree/eb5f1663ed0743660b8aa749a43f35f505baa325	迁移完成后进行如下修改。
			• 修改./model/models.py的create_module函数。
			修改前：
			`module_list[j][0].bias = mindspore.Parameter(bias_, …)`
			修改后：
			`module_list[j][0].bias = mindspore.Parameter(bias.reshape(bias_.shape), …)`
			• 修改./utils/datasets.py
			修改前：
			`if os.path.isfile(cache_path):`
			修改后：
			`if False:`
			• 对./utils/loss.py的build_targets函数：
			修改前
			`gij = x2ms_adapter.tensor_api.long((…))`
			`gi, gj = gij.T`
			`…`
			`tbox.append(…)`
			修改后
			`gij = x2ms_adapter.tensor_api.long((…)).T`
			`gi, gj = gij`
			`…`
			`gij = gij.T`
			`tbox.append(…)`
			• 对./train.py，
			– 把**if '.bias' in k:** 修改为**if '.bias' in k or '.beta' in k:**
			– 字符串**'Conv2d.weight'**修改为**'.weight'**
			• 运行多卡场景下，将./train.py中“testloader = create_dataloader(… ）”的rect参数修改为False。

206	YOLOv4-tiny		
207	YOLOv4-pacsp		
208	YOLOv4-paspp		
209	YOLOv4-csp-leaky		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>210</td><td>YOLOv5l</td><td rowspan="5">https://github.com/ultralytics/yolov5/tree/8c420c4c1fb3b83ef0e60749d46bcc2ec9967fc5</td><td rowspan="5">迁移完成后进行如下修改。
• 对./models/yolo.py：
class Detect(mindspore.nn.Cell):
stride = None #删除
…
def __init__(self, …):
…
self.stride = None #新增
• 对./utils/loss.py的build_targets函数：
修改前
gij = x2ms_adapter.tensor_api.long((…))
gi, gj = gij.T
…
tbox.append(…)
修改后
gij = x2ms_adapter.tensor_api.long((…)).T
gi, gj = gij
…
gij = gij.T
tbox.append(…)
• 对./val.py中run函数：
– 删除“path, shape = Path(paths[si]), shapes[si][0]”。
– 删除scale_coords函数的调用处。
– 删除“callbacks.run('on_val_image_end', …)”。
• 将./models/目录下对应模型配置文件{model_name}.yaml中所有nn.*修改为x2ms_adapter.torch_api.nn_api.nn.*。
• 运行多卡场景下，将./train.py中“val_loader = create_dataloader(…）”的rect参数修改为False。</td></tr>
<tr><td>211</td><td>YOLOv5m</td></tr>
<tr><td>212</td><td>YOLOv5n</td></tr>
<tr><td>213</td><td>YOLOv5s</td></tr>
<tr><td>214</td><td>YOLOv5x</td></tr>
<tr><td>215</td><td>YOLOX</td><td>https://github.com/bubbliiiing/yolox-pytorch/tree/1448e849ac6cdd7d1cec395e30410f49a83d44ec</td><td>迁移后，修改如下内容。
• 注释./train.py第341行代码。
#'adam' : optim_register.adam(pg0, Init_lr_fit, betas = (momentum, 0.999))
• 进行训练前，防止hccl超时，执行如下命令。
export HCCL_CONNECT_TIMEOUT=3000</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注
216	AAGCN-ABSA	https://git.openi.org.cn/PCLNLP/SentimentAnalysisNLP/src/commit/7cf38449dad742363053c4cc380ebfe33292184d	-
217	CAER-ABSA		• 依赖三方库pytorch-pretrained-bert，下载并将其子目录pytorch_pretrained_bert拷贝至SentimentAnalysisNLP/目录下。 • 将./SentimentAnalysisNLP/pytorch_pretrained_bert/modeling.py中158行的BertLayerNorm类定义移至try-except语块外。
218	GIN-ABSA		• 依赖三方库pytorch-pretrained-bert，下载并将其子目录pytorch_pretrained_bert拷贝至SentimentAnalysisNLP/目录下。 • 由于MindSpore中不支持在数据处理过程中创建Tensor，需要在./GIN-ABSA/data_utils.py中去除数据集初始化中的tensor创建，包括：去除147行的torch.tensor()操作和234行的torch.tensor()操作。
219	Scon-ABSA		• 依赖huggingface中bert-base-uncased的预训练权重，需要下载pytorch_model.bin，转换成MindSpore格式的pytorch_model.ckpt后，在脚本中加载该转换后的模型权重。 • 依赖三方库pytorch-pretrained-bert，下载并将其子目录pytorch_pretrained_bert拷贝至SentimentAnalysisNLP/目录下。 • 将./pytorch_pretrained_bert/modeling.py中158行的BertLayerNorm类定义移至try-except语块外。
220	Trans-ECE		• 依赖huggingface中bert-base-chinese的预训练权重，需要下载到当前目录的bert-base-chinese下，并将模型权重pytorch_model.bin转换到MindSpore格式的pytorch_model.ckpt。 • 由于原始代码中存在缺陷，需要将./Trans-ECE/Run.py中48、49行使用list包裹filter，并删除54行多余的trans_optimizer参数。 • 由于当前不支持自定义优化器，需要将./Trans-ECE/Run.py中55行的BertAdam改为optim.Adam优化器。

序号	模型	原始训练工程代码链接参考	备注
221	PyramidNet 101	https://github.com/dyhan0920/PyramidNet-PyTorch/tree/5a0b32f43d79024a0d9cd2d1851f07e6355daea2	迁移前，进行如下修改。 • 由于原始仓库代码对于python版本和pytorch版本有限制，需要根据https://github.com/dyhan0920/PyramidNet-PyTorch/issues/5进行适配性修改。 • 由于迁移后代码不需要torchvision模块，需要注释train.py的23~25行。 #model_names = sorted(name for name in models.__dict__ # if name.islower() and not name.startswith("__") # and callable(models.__dict__[name]))
222	PyramidNet 164 bottleneck		
223	PyramidNet 200 bottleneck		

表 4-3 TensorFlow 2 模型列表

序号	模型	原始训练工程代码链接参考	备注

1	ALBERT_base_v2	https://github.com/huggingface/transformers/tree/49cd736a28	• 迁移前，需要把原仓库的模板文件移走，这些文件本质不是python文件却以.py后缀命名。
			`mv templates ../`
			• 迁移后，请进行以下修改：
			– 对./examples/tensorflow/language-modeling/run_mlm.py：
			– 补充以下导包语句：
			`from x2ms_adapter.keras.losses import SparseCategoricalCrossentropy`
			– 将DataCollatorForLanguageModeling参数return_tensors的值"tf"改为"np"：
			修改前：
			`data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=data_args.mlm_probability, return_tensors="tf")`
			修改后：
			`data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=data_args.mlm_probability, return_tensors="np")`
			– 修改model.compile调用的参数：
			修改前：
			`model.compile(optimizer=optimizer)`
			修改后：
			`model.compile(optimizer=optimizer, loss=SparseCategoricalCrossentropy(True))`
			– 修改./src/transformers/modeling_tf_utils.py的dtype_byte_size方法：
			修改前：
			`bit_search = re.search("[^\d](\d+)$", dtype.name)`
			修改后：
			`bit_search = re.search("[^\d](\d+)$", str(dtype))`

2	ALBERT_large_v2		
3	ALBERT_xlarge_v2		
4	ALBERT_xxlarge_v2		
5	ALBERT_base_v1		
6	ALBERT_large_v1		
7	ALBERT_xlarge_v1		
8	ALBERT_xxlarge_v1		
9	roberta-base		
10	roberta-large		
11	RBT6		
12	RBT4		
13	RBTL3		
14	RBT3		

序号	模型	原始训练工程代码链接参考	备注

15	CIFAR-VGG	https://github.com/dragen1860/TensorFlow-2.x-Tutorials/tree/9861a308283d17693d497fdfaab20b92c0b26d08	• 为减少不必要的迁移时间，建议选择仅迁移06-CIFAR-VGG文件夹。
			• 下载数据集**CIFAR-10 python version**并解压处理。
			• 迁移后修改./06-CIFAR-VGG_x2ms/main.py文件如下内容：
			1. 修改数据集路径，示例如下：
			修改前：
			`(x,y), (x_test, y_test) = x2ms_ks_cifar10.load_data()`
			修改后：
			`(x,y), (x_test, y_test) = x2ms_ks_cifar10.load_data(data_dir='`***数据集路径***`')`
			2. 删除map操作。
			修改前：
			`train_loader = train_loader.map(prepare_cifar).shuffle(50000).batch(256)`
			`test_loader = test_loader.map(prepare_cifar).shuffle(10000).batch(256)`
			修改后：
			`train_loader = train_loader.shuffle(50000).batch(256)`
			`test_loader = test_loader.shuffle(10000).batch(256)`
			3. 指定损失函数的**reduction**。
			修改前：
			`criteon = x2ms_ks_losses.CategoricalCrossentropy(from_logits=True)`
			修改后：
			`criteon = x2ms_ks_losses.CategoricalCrossentropy(from_logits=True, reduction='mean')`

16	DenseNet_121	https://github.com/calmisential/Basic_CNNs_TensorFlow2/tree/f063c84451f12e904f9c91c51278be52afccb0c2	• 请据需要，在./configuration.py中进行epoch、batch_size、数据集路径等配置。 • 迁移前，将./models/__init__.py文件的regnet.RegNet代码行注释。 `#regnet.RegNet()`
17	DenseNet_169		
18	EfficientNet_B0		
19	EfficientNet_B1		
20	Inception_V4		
21	MobileNet_V1		

序号	模型	原始训练工程代码链接参考	备注
22	MobileNet_V2		
23	MobileNet_V3_Large		
24	MobileNet_V3_Small		
25	ResNet_101		
26	ResNet_152		
27	ResNet_18		
28	ResNet_34		
29	ResNet_50		
30	ResNext_101		
31	ResNext_50		
32	Shufflenet_V2_x0_5		
33	Shufflenet_V2_x1_0		

序号	模型	原始训练工程代码链接参考	备注
34	EfficientNet_B2	https://github.com/calmisential/Basic_CNNs_TensorFlow2/tree/f063c84451f12e904f9c91c51278be52afccb0c2	• 迁移前参照reademe对数据集进行处理。 • 请根据需要，在./configuration.py中进行epoch、batch_size、数据集路径、分类数等配置。 • 修改./configuration.py中的第7行，如数据集类别为10，进行以下修改。 修改前： NUM_CLASSES = 5 修改后： NUM_CLASSES = 10 • 迁移前，将./models/__init__.py文件21行注释。 #regnet.RegNet() • DenseNet_201和DenseNet_264模型在迁移前，在./models/__init__.py文件第20行**shufflenet_v2.shufflenet_2_0x()**, 后添加如下代码。 densenet.densenet_201(), densenet.densenet_264()
35	EfficientNet_B3		
36	EfficientNet_B4		
37	EfficientNet_B5		
38	EfficientNet_B6		
39	EfficientNet_B7		
40	SE_ResNet_50		
41	SE_ResNet_101		
42	SE_ResNet_152		
43	SE_ResNext_50		
44	SE_ResNext_101		
45	DenseNet_201		
46	DenseNet_264		
47	AFM	https://github.com/ZiyaoGeng/Recommender-System-with-TF2.0/tree/1d2aa5bf551873d5626539c196705db46d55c7b6	各个网络文件夹均依赖./data_process/目录，请直接迁移Recommender-System-with-TF2.0/目录或将./data_process/复制至网络文件夹下后再进行迁移。
48	Caser		
49	DCN		
50	Deep_Crossing		
51	DeepFM		
52	DNN		
53	FFM		
54	FM		

序号	模型	原始训练工程代码链接参考	备注
55	MF		
56	NFM		
57	PNN		
58	WDL		

59	BiLSTM-CRF	https://github.com/kangyishuai/BiLSTM-CRF-NER/tree/84bde29105b13cd8128bb0ae5d043c4712a756cb	• 训练需在MindSpore 1.7版本中执行。
			• 根据原训练工程README.md下载完整数据集，解压数据集并将里面的文件拷贝至./data中。
			• 迁移后，视训练情况，将./main.py中的batch_size，hidden_num和embedding_size参数值适当调小，如以下示例：
			params = {
			"maxlen": 128,
			"batch_size": **140**,
			"hidden_num": **64**,
			"embedding_size": **64**,
			"lr": 1e-3,
			"epochs": 10
			}

60	FCN	https://github.com/YunYang1994/TensorFlow2.0-Examples/tree/299fd6689f242d0f647a96b8844e86325e9fcb46/5-Image_Segmentation/FCN	./parser_voc.py中使用的scipy.misc.imread方法为scipy 1.2.0以前的旧版本API，mindspore最低兼容scipy 1.5.2，因此请使用scipy的官方弃用警告中推荐的imageio.imread。

61	GoogleNet	https://github.com/marload/ConvNets-TensorFlow2/tree/29411e941c4aa72309bdb53c67a6a2fb8db57589	• 迁移后的load_data()接口需要通过data_dir参数指定数据集路径或将数据集放置在默认路径~/x2ms_datasets/cifar10/cifar-10-batches-py下。
			• Vgg16模型需要在迁移前修改./utils.py 中第9行。
			修改前：
			if nets_name == 'VGG16':
			修改后：
			if nets_name == 'vgg16':

62	SqueezeNet		
63	Vgg11		
64	Vgg13		
65	Vgg16		
66	Vgg19		

序号	模型	原始训练工程代码链接参考	备注
67	Unet	https://github.com/YunYang1994/TensorFlow2.0-Examples/tree/299fd6689f242d0f647a96b8844e86325e9fcb46/5-Image_Segmentation/Unet	数据集请使用Membrane，可从该训练工程的README.md中获取。

68	U-Net_Med	https://github.com/monchhichizzq/Unet_Medical_Segmentation/tree/876d5adbeb17fa79dbf644dd1d6e91840f904e2d	• 迁移前需要修改./Preprocess/Data_Loader.py文件。
			1. 11行新增导入如下内容：
			`import random`
			2. 修改shuffle 函数为random.shuffle。
			修改前：
			`shuffle(self.train_lines)`
			修改后：
			`random.shuffle(self.train_lines)`
			• 迁移后，下载数据集**Medical_Datasets**，并将数据集放在迁移后的模型根目录下。
			• 迁移后，修改./train.py文件，指定使用的优化器。
			修改前：
			`optimizer=x2ms_ks_optimizers.Adam(lr=lr),`
			修改后：
			`optimizer=x2ms_ks_optimizers.SGD(lr=lr),`

69	VAE	https://github.com/dragen1860/TensorFlow-2.x-Tutorials/tree/9861a308283d17693d497fdfaab20b92c0b26d08	为减少不必要的迁移时间，建议选择仅迁移12-VAE文件夹。
			迁移后需要进行如下修改：
			• 下载数据集**Fashion-MNIST**。
			• 修改./12-VAE_x2ms/main.py文件。
			1. 删除如下内容。
			`assert tf.__version__.startswith('2.')`
			2. 指定数据集路径。
			`(x_train, y_train), (x_test, y_test) = x2ms_ks_mnist.load_fashion_data('`***数据集路径***`')`
			3. 修改原模型中训练的输入。
			修改前：
			`one_step_cell()`
			修改后：
			`one_step_cell(x)`

序号	模型	原始训练工程代码链接参考	备注
70	Vit	https://github.com/tuvovan/Vision_Transformer_Keras/tree/6a1b0959a2f5923b1741335aca5bc2f8dcc7c1f9	• 迁移后的load()接口需要通过data_dir参数指定数据集路径或将数据集放置在默认路径~/x2ms_datasets/cifar10/cifar-10-batches-bin下。 • 需要删除train.py中“early_stop = tf.keras.callbacks.EarlyStopping(patience=10),”中的逗号以保证callback对象为单一的实例而非元组。

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>71</td><td>Yolov5-l-mish</td><td rowspan="4">https://github.com/LongxingTan/tfyolo/tree/df4fa04aa9ee10cb8147f04f63f1484a1fa926fa</td><td rowspan="4">1. 根据原网络ReadMe.md修改并处理数据集。
2. 迁移前需要进行如下修改。
a. 在./yolo/model/yolo.py第11行新增如下内容。
from tensorflow.keras.layers import Layer, Conv2D
b. 修改./yolo/dataset/load_data.py中第30-31行。
修改前：
dataset = dataset.map(self.transform, num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.batch(batch_size).prefetch(tf.data.experimental.AUTOTUNE)
修改后：
dataset = dataset.map(self.transform, output_nums=4)
dataset = dataset.batch(batch_size)
c. 修改./yolo/dataset/load_data.py中第36行。
修改前：
return image, label_encoder
修改后：
return (image,) + label_encoder
3. 执行迁移操作。
4. 迁移后需要执行如下修改。
a. 修改./yolo/train.py中第106行。
修改前：
for step, (image, target) in enumerate(train_dataset):
修改后：
for step, (image, *target) in enumerate(train_dataset):
b. 修改./yolo/train.py中第145行。
修改前：
self.optimizer.lr.assign(lr)
修改后：
self.optimizer.learning_rate = lr
c. 修改./yolo/train.py中第178行。
修改前：
trainer.anchors,
修改后：
trainer.anchors.asnumpy(),
d. 修改./yolo/dataset/label_anchor.py中第7行。
修改前：
from x2ms_adapter import ops as x2ms_ops
修改后：</td></tr>
<tr><td>72</td><td>Yolov5-m-mish</td></tr>
<tr><td>73</td><td>Yolov5-s-mish</td></tr>
<tr><td>74</td><td>Yolov5-x-mish</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注
			from x2ms_adapter import numpy_ops as x2ms_ops e. 修改./yolo/model/module.py中第193行。 修改前： return x2ms_transforms.resize_image(x, (x2ms_ops.get_shape(x)[1] * self.ratio, x2ms_ops.get_shape(x)[2] * self.ratio), method=self.method) 修改后： return x2ms_ops.to_tensor(x2ms_transforms.resize_image(x, (x2ms_ops.get_shape(x)[1] * self.ratio, x2ms_ops.get_shape(x)[2] * self.ratio), method=self.method)) f. 仅**Yolov5-s-mish**网络需执行该步骤。 修改./yolo/model/yolo.py的类Detect中的stride值，修改第88行。 修改前： self.stride = np.array([8, 16, 32], np.float32) 修改后： self.stride = np.array([16, 32, 64], np.float32) g. 仅**Yolov5-x-mish**网络需执行该步骤。 修改./yolo/configs/yolo-x-mish.yaml的第4行。 修改前： width_multiple: 1.25 修改后： width_multiple: 1.0 5. 执行迁移后训练。 ./yolo/configs/路径下存放4个模型的配置文件，执行训练任务时，需要注意用“--yaml_dir”参数指定不同模型对应的yaml文件。 启动任务的示例命令如下： python3 yolo/train.py \ --train_annotations_dir data/voc/voc_train.txt \ --test_annotations_dir data/voc/voc_test.txt \ --class_name_dir data/voc/voc.names \ --yaml_dir yolo/configs/yolo-l-mish.yaml \ --n_epochs 5

表 4-4 TensorFlow 1 模型列表

序号	模型	原始训练工程代码链接参考	备注

1	ALBERT-base-v2	https://github.com/google-research/ALBERT/tree/a36e095d3066934a30c7e2a816b2eeb3480e9b87	迁移前，需要进行以下修改：
			• 在./classifier_utils.py，将以下语句
			`if t.dtype == tf.int64:`
			修改为
			`if t.dtype == "int64":`
			• 在./optimization.py，进行以下修改：
			– 修改前：
			`optimizer = AdamWeightDecayOptimizer(`
			修改后：
			`optimizer = tf.keras.optimizers.Adam(`
			– 修改前：
			`train_op = tf.group(train_op, [global_step.assign(new_global_step)])`
			修改后：
			`train_op = tf.group(train_op, global_step)`
			• 数据集使用Glue-MNLI时，record数据集需参考README自行生成。

2	ALBERT-large-v2		
3	ALBERT-xlarge-v2		
4	ALBERT-xxlarge-v2		

序号	模型	原始训练工程代码链接参考	备注
5	AFM	https://github.com/cheungdaven/DeepRec/tree/68a34cb495911e797d85cbd962526188f4aede12	迁移后需要进行以下修改： • 使用**NNMF、NRR、I-AutoRec、U-AutoRec、FM、NFM、AFM、DEEP-FM**模型时，需要根据实际情况修改数据集所在路径，示例如下： 1. 修改./test/test_rating_pred.py文件中的数据集路径，示例如下。 修改前： path="../Data/ml100k/movielens_100k.dat" 修改后： path="./data/ml100k/movielens_100k.dat" 2. 修改./utils/load_data/load_data_content.py文件中的数据集路径，示例如下： 修改前： train_file = "../Data/frappe/frappe.train.libfm" test_file = "../Data/frappe/frappe.test.libfm" 修改后： train_file = "./data/frappe/frappe.train.libfm" test_file = "./data/frappe/frappe.test.libfm" • 使用**Caser、PRME、AttRec**模型时，需要依据实际情况修改./test/testSeqRec.py文件中数据集路径，示例如下： 修改前： "../data/ml100k/temp/train.dat" "../data/ml100k/temp/test.dat" 修改后： "./data/ml100k/temp/train.dat" "./data/ml100k/temp/test.dat" • 使用**NRR**模型时，修改./models/rating_prediction/nrr.py文件。 修改前： input = x2ms_ops.matmul(user_latent_factor, W_User) + x2ms_ops.matmul(item_latent_factor, W_Item) + b 修改后： input = x2ms_ops.matmul(user_latent_factor, W_User) + x2ms_ops.matmul(item_latent_factor, W_Item) + b.value • 使用**AFM**模型时，修改./models/rating_prediction/afm.py文件。 1. 修改前： self.train_features = x2ms_base_v1.placeholder(mstype.int32, shape=[None,None]) self.y = x2ms_base_v1.placeholder(mstype.float32, shape=[None, 1]) 修改后： self.train_features = x2ms_base_v1.placeholder(mstype.int32, shape=[valid_dim, valid_dim])
6	AttRec		
7	Caser		
8	DEEP-FM		
9	FM		
10	I-AutoRec		
11	NFM		
12	NNMF		
13	NRR		
14	PRME		
15	U-AutoRec		

序号	模型	原始训练工程代码链接参考	备注
			self.y = x2ms_base_v1.placeholder(mstype.float32, shape=[valid_dim, 1]) 2. 修改前： self.attention_relu = x2ms_ops.reduce_sum(x2ms_ops.multiply(self.attention_p, x2ms_ops.relu(self.attention_mul + self.attention_b)), 2, keep_dims=True) 修改后： self.attention_relu = x2ms_ops.reduce_sum(x2ms_ops.multiply(self.attention_p, x2ms_ops.relu(self.attention_mul + self.attention_b.value)), 2, keep_dims=True) ● 使用**Caser**模型时，修改./models/seq_rec/Caser.py文件。 1. 根据实际情况，适当修改**learning_rate**和**epoch**的值，修改后示例如下： def __init__(self, sess, num_user, num_item, learning_rate=0.00001, reg_rate=1e-2, epoch=100, batch_size=20000, show_time=False, T=1, display_step=1000, verbose=False): 2. 修改前： return params[0] 修改后： return params ● 使用**PRME**模型时，修改./models/seq_rec/PRME.py文件。 1. 根据实际情况，适当修改**learning_rate**、**epoch**和**batch_size**的值，修改后示例如下。 def __init__(self, sess, num_user, num_item, learning_rate=0.00001, reg_rate=1e-2, epoch=100, batch_size=20000, show_time=False, T=1, display_step=1000, verbose=False): 2. 修改前： return - self.sess.run([self.test_prediction], feed_dict={self.user_id: user_id, self.item_seq: self.test_sequences[user_id, :], self.item_id_test: item_id})[0] 修改后： return - self.sess.run([self.test_prediction], feed_dict={self.user_id: user_id, self.item_seq: self.test_sequences[user_id, :], self.item_id_test: item_id}) ● 因算子差异，使用**AttRec**模型时需修改./models/seq_rec/AttRec.py文件。 修改前： signal = x2ms_ops.pad(signal, [[0, 0], [0, x2ms_ops.mod(channels, 2)]]) 修改后： signal = x2ms_ops.pad(signal, [[0, 0], [0, int(x2ms_ops.mod(channels, 2))]])

序号	模型	原始训练工程代码链接参考	备注
16	Attention-Based Bidirectional RNN	https://github.com/dongjun-Lee/text-classification-models-tf/tree/768ea13547104f56786c52f0c6eb99912c816a09	由于dropout算子在mindspore中已经对training参数做了处理，所以需要将模型定义文件中self.keep_prob属性直接修改为0.5，无需通过where判断。
17	Character-level CNN		
18	RCNN		
19	Very Deep CNN		
20	Word-level Bidirectional RNN		
21	Word-level CNN		

<table>
<tr><th>序号</th><th>模型</th><th>原始训练工程代码链接参考</th><th>备注</th></tr>
<tr><td>22</td><td>BERT-Tiny</td><td rowspan="5">https://github.com/google-research/bert/tree/eedf5716ce1268e56f0a50264a88cafad334ac61</td><td rowspan="5">迁移前，请进行如下修改：
● ./run_classifier.py文件中：
– 将hidden_size = output_layer.shape[-1].value的“.value”删除（源代码592行）。
hidden_size = output_layer.shape[-1]
– 注释以下代码（源代码869、870行）：
#file_based_convert_examples_to_features(
train_examples, label_list, FLAGS.max_seq_length, tokenizer, train_file)
– 在_decode_record函数中，将源代码529行
if t.dtype == tf.int64:
改为
if t.dtype == 'int64’
● 对./optimization.py：
– 将AdamWeightDecayOptimizer的实例化代码替换成optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
#optimizer = AdamWeightDecayOptimizer(
#…
#)
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
– 修改前：
train_op = tf.group(train_op, [global_step.assign(new_global_step)])
修改后：
train_op = tf.group(train_op, global_step)</td></tr>
<tr><td>23</td><td>BERT-Mini</td></tr>
<tr><td>24</td><td>BERT-Small</td></tr>
<tr><td>25</td><td>BERT-Medium</td></tr>
<tr><td>26</td><td>BERT-Base</td></tr>
<tr><td>27</td><td>Inception v1</td><td rowspan="4">https://github.com/tensorflow/models/tree/164bab98cc218f5c8cbd6ec1156cd6f364032a1b</td><td rowspan="4">迁移后在./models/research/slim_x2ms/train_image_classifier.py文件中做如下修改。
1. image_preprocessing_fn函数需要添加入参crop_image=False。例如：
image = image_preprocessing_fn(image, train_image_size, train_image_size, crop_image=False)
2. 在train_tensor = x2ms_ops.identity(total_loss, name='train_op')语句前，新增如下内容。
total_loss = x2ms_ops.depend(total_loss, update_op)</td></tr>
<tr><td>28</td><td>Inception v2</td></tr>
<tr><td>29</td><td>Inception v4</td></tr>
<tr><td>30</td><td>Inception-ResNet-v2</td></tr>
</table>

序号	模型	原始训练工程代码链接参考	备注

31	RBT6	https://github.com/bojone/bert4keras/tree/9c1c916def4d515a046c414	迁移前，需进行以下修改：
			• 在./examples/task_language_model.py中：
			– 修改checkpoint_path，config_path，dict_path，输入的训练数据，batch_size的值。
			– `txt = open(txt, encoding='gbk').read()` 改为 `txt = open(txt, encoding='utf8').read()`
			• 对./bert4keras/layers.py，在from keras.layers import *语句后增加以下导入语句：
			`from keras.layers import Input, Dropout, Lambda, Add, Dense, Activation`
			• 对./bert4keras/models. py，在from bert4keras.layers import *语句后增加以下导入语句：
			`from bert4keras.layers import Input, Dropout, Lambda, Add, K, Dense, Activation`

32	RBT4		
33	RBTL3		
34	RBT3		
35	RoBERTa-wwm-ext-large		
36	RoBERTa-wwm-ext		

37	Bi-LSTM-CRF	https://github.com/fzschornack/bi-lstm-crf-tensorflow/tree/5181106	• 迁移前需要新建./bi-lstm-crf-tensorflow.py文件，并将bi-lstm-crf-tensorflow.ipynb中的代码拷贝至该新建的Python文件中。
			• 迁移后，视训练情况修改./bi-lstm-crf-tensorflow.py中num_units变量的赋值语句值。以修改为64为例：
			`#num_units = 128`
			`**num_units = 64**`

38	CNN-LSTM-CTC	https://github.com/watsonyanghx/CNN_LSTM_CTC_Tensorflow/tree/6999cd19285e7896cfe77d50097b0d96fb4e53e8	• 迁移前，将utils.py中的43行注释
			`#tf.app.flags.DEFINE_string('log_dir', './log', 'the logging dir')`
			• 迁移后，适当调小./utils.py中validation_steps的值，以便训练时快速观察模型收敛效果，以修改为50为例：
			`x2ms_FLAGS.define_integer('validation_steps', **50**, 'the step to validation')`
			• 在工程根目录下，创建./imgs/train, ./imgs/val文件夹，放入指定训练、测试数据。

序号	模型	原始训练工程代码链接参考	备注

39	DCN	https://github.com/princewen/tensorflow_practice/tree/master/recommendation	迁移前需要对DCN模型做如下修改。
			1. 修改DCN.py文件63行。
			修改前：
			`self.numeric_value = tf.placeholder(tf.float32, [None,None],name='num_value')`
			修改后：
			`self.numeric_value = tf.placeholder(tf.float32, [None,9],name='num_value')`
			2. 修改DCN.py文件89行。
			修改前：
			`self.y_deep = x2ms_nn.dropout(self.y_deep,self.dropout_keep_deep[i+1])`
			修改后：
			`self.y_deep = x2ms_nn.dropout(self.y_deep,self.dropout_keep_deep[0])`
			3. 修改DCN.py文件96、97行。
			修改前：
			`x_l = tf.tensordot(tf.matmul(self._x0, x_l, transpose_b=True), self.weights["cross_layer_%d" % l],1) + self.weights["cross_bias_%d" % l] + x_l`
			修改后：
			`x_l = self.weights["cross_bias_%d" % l] + tf.tensordot(tf.matmul(self._x0, x_l, transpose_b=True), self.weights["cross_layer_%d" % l],1) + x_l`
			4. 注释DCN.py文件142行。
			`#self.sess.run(init)`
			5. 修改DCN.py文件150行。
			修改前：
			`variable_parameters *= dim.value`
			修改后：
			`variable_parameters *= dim`

40	MLR5		• 迁移前，自行下载adults数据集，将里面的文件拷贝至./Basic-MLR-Demo/data目录下（请自行创建data目录）。 • 迁移前，根据运行的模型类型，将mlr.py文件中的**m=2**修改为**m=5**或**m=10**或**m=15**或**m=20**。
41	MLR10		
42	MLR15		
43	MLR20		

序号	模型	原始训练工程代码链接参考	备注

44	PNN		迁移前需要对PNN模型做如下修改。
			1. 修改PNN.py文件109行。
			修改前：
			`self.y_deep = x2ms_nn.dropout(self.y_deep,self.dropout_keep_deep[i+1])`
			修改后：
			`self.y_deep = x2ms_nn.dropout(self.y_deep,self.dropout_keep_deep[0])`
			2. 注释PNN.py文件141行。
			`#self.sess.run(init)`
			3. 修改PNN.py文件149行。
			修改前：
			`variable_parameters *= dim.value`
			修改后：
			`variable_parameters *= dim`
			4. 修改main.py 文件100行。
			修改前：
			`y_train_pnn, y_test_pnn = run_base_model_pnn(dfTrain, dfTest, folds, pnn_params)`
			修改后：
			`run_base_model_pnn(dfTrain, dfTest, folds, pnn_params)`

45	LeNet	https://github.com/Jackpopc/aiLearnNotes/tree/7069a705bbcbea1ac24	• 迁移后，自行下载MNIST数据集，放至执行命令的目录下。解压MNIST里的所有.gz文件并删除原gz文件。 • 若网络收敛不佳，可尝试调小学习率LR、增大训练周期EPOCHS。
46	AlexNet		

序号	模型	原始训练工程代码链接参考	备注
47	ResNet-18	https://github.com/taki0112/ResNet-Tensorflow/tree/f395de3a53d	迁移前需要安装jedi依赖。 迁移后需要做以下适配。 • 迁移后的load()接口需要通过data_dir参数指定数据集路径或将需要的数据集放置在默认路径： – ~/x2ms_datasets/cifar100/cifar-100-python – ~/x2ms_datasets/cifar10/cifar-10-batches-py – ~/x2ms_datasets/mnist.npz – ~/x2ms_datasets/fashion-mnist • 由于MindSpore不支持一次性做大批量数据的测试。 – 需要自行修改代码将测试集分批次做测试。 – 测试集的placeholder的shape第一维改为1。
48	ResNet-34		
49	ResNet-50		
50	ResNet-101		
51	ResNet-152		

4.2.4.2 FAQ

4.2.4.2.1 保存 Checkpoint 文件成功，但找不到该文件，可能是哪些原因?

问题描述

保存Checkpoint文件后，对该文件进行操作时，显示该文件不存在。

```
Traceback (most recent call last):
  File "main.py", line 470, in <module>
    main()
  File "main.py", line 138, in main
    main_worker(args.gpu, ngpus_per_node, args)
  File "main.py", line 281, in main_worker
    }, is_best)
  File "main.py", line 376, in save_checkpoint
    shutil.copyfile(filename, 'model_best.pth.tar')
  File "/usr/local/python3.7.5/lib/python3.7/shutil.py", line 120, in copyfile
    with open(src, 'rb') as fsrc:
FileNotFoundError: [Errno 2] No such file or directory: 'checkpoint.pth.tar'
```

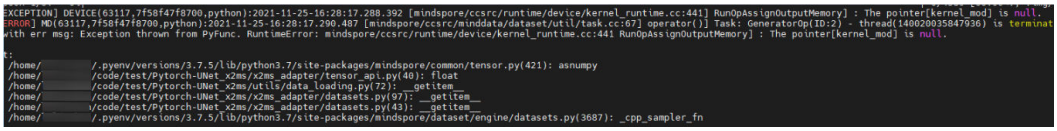
解决方法

MindSpore中load_checkpoint和save_checkpoint接口仅支持“.ckpt”后缀的文件，当文件名为非“.ckpt”后缀时，会自动加上“.ckpt”后缀，因此实际保存的文件名可能和传入的文件名不一致，此时需要用户将传入文件名末尾加上“.ckpt”。

4.2.4.2.2 数据处理时出现错误，可能是哪些原因?

问题描述

数据处理时出现如下错误。



解决方法

在MindSpore中，数据处理过程中需要保持中间处理结果为numpy array，且数据处理过程中不建议与MindSpore网络计算的算子混合使用。请检查数据处理中是否包含转换为Tensor和使用MindSpore网络算子的操作，若有，请使用numpy中的等效操作替换。

4.2.4.2.3 运行迁移后代码报未定义 API 的错误

问题描述

运行迁移后代码报未定义API的错误，该API出现于不支持API列表unsupported_api.csv且不是迁移前代码中的API。

问题原因

libcst0.4.3以前的版本对形如“from a import **b**, **b**c”的代码有解析问题。例如，对于**from keras.datasets import cifar10, cifar100, mnist, fashion_mnist**，cifar10是cifar100的前缀，分析迁移工具将无法解析该API。

解决方法

升级libcst版本至0.4.3后重新进行脚本迁移。

4.3 PyTorch GPU2Ascend

4.3.1 概述

昇腾NPU是AI算力的后起之秀，但目前训练和在线推理脚本大多是基于GPU的。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。

PyTorch GPU2Ascend工具提供了将基于GPU的脚本迁移为基于NPU的脚本的自动化方法，节省了人工手动进行脚本迁移的学习成本与工作量，大幅提升了迁移效率。同时提供分析方法，帮助用户分析PyTorch训练脚本的算子支持情况。当前支持1.11.0、2.0.1、2.1.0版本的训练脚本分析和迁移。

约束说明

- PyTorch GPU2Ascend工具迁移后脚本的执行逻辑与迁移前保持一致。
- 待迁移脚本需要在GPU环境下且基于Python3.7及以上能够跑通。

- 若原始代码中调用了三方库，迁移过程可能会存在适配问题。在迁移原始代码前，用户需要根据已调用的三方库，自行安装昇腾已适配的三方库版本，已适配的三方库信息和使用指南请参考《PyTorch网络迁移和训练指南》中“模型套件和第三方库”章节。
- APEX中使用的FusedAdam优化器不支持使用工具进行迁移，若原始代码中包含该优化器，用户需自行修改。

4.3.2 前提条件

分析和迁移前须安装如下依赖。

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

```
pip3 install pandas        #pandas版本需大于或等于1.2.4
pip3 install libcst        #语义分析库，用于解析Python文件
pip3 install prettytable   #将数据可视化为图表形式
pip3 install jedi          #可选，用于跨文件解析，建议安装
```

4.3.3 分析操作

PyTorch GPU2Ascend提供脚本分析功能，帮助用户在执行迁移操作前，分析PyTorch训练脚本的算子和API支持情况，并输出训练脚本中API精度和性能调优的专家建议。

操作步骤

步骤1 通过以下任一方式启动分析任务。

- 单击工具栏中图标。
- 在菜单栏选择“Ascend > Migration Tools > PyTorch GPU2Ascend”。
- 右键单击训练工程，然后选择“PyTorch GPU2Ascend”。

步骤2 参数配置。

默认进入“Analysis”页面如图4-3所示，用户自行根据实际情况配置参数。

图 4-3 Analysis 参数配置

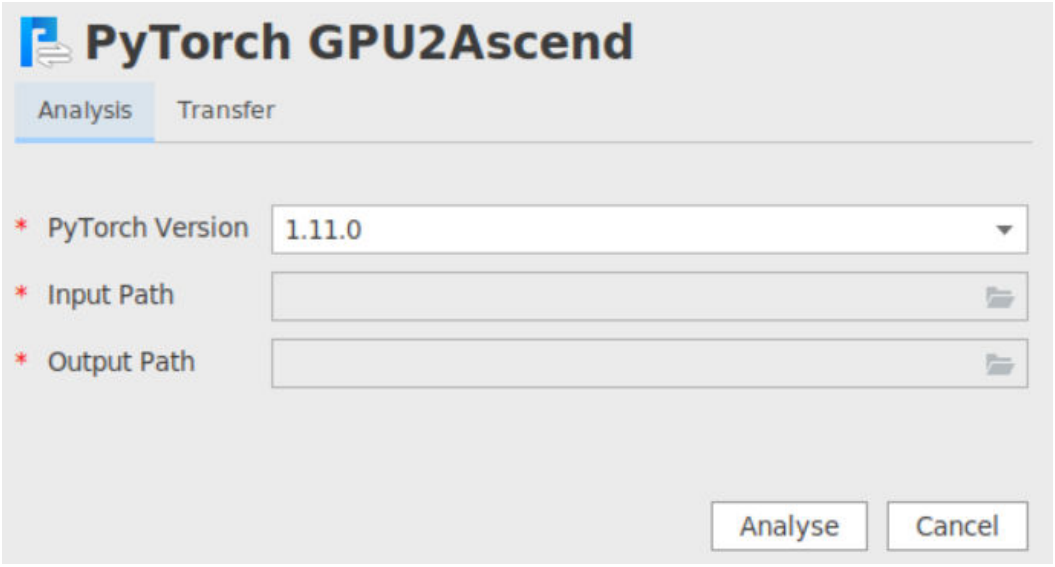


表 4-5 PyTorch GPU2Ascend 参数说明

参数	参数说明
PyTorch Version	待分析脚本的PyTorch版本。目前支持1.11.0、2.0.1、2.1.0。 必选。默认为1.11.0。
Input Path	待分析脚本文件所在目录。单击文件夹图标选择目录。 必选。
Output Path	分析结果文件输出路径。单击文件夹图标选择目录。 必选。

步骤3 单击“Analyse”，执行分析任务。

完成后，**Output Path**输出目录下查看结果文件。

```
├── xxx_analysis     // 分析结果输出目录
│   ├── cuda_op_list.csv          //cuda算子列表
│   ├── unknown_api.csv           //支持情况存疑的API列表
│   ├── unsupported_api.csv       //不支持的API列表
│   ├── api_precision_advice.csv    //API精度调优的专家建议
│   ├── api_performance_advice.csv  //API性能调优的专家建议
│   ├── pytorch_analysis.txt      // 分析过程日志
```

对于当前框架不支持的API，可以在**昇腾开源社区**寻求帮助。

用户可以参考精度和性能调优的专家建议，除此之外还可以使用**9 精度比对**和**10 性能分析**工具进行调优。

----结束

4.3.4 迁移操作

PyTorch GPU2Ascend提供脚本分析和迁移功能，帮助用户在执行分析迁移操作中，分析PyTorch训练脚本的算子和API支持情况，并将其迁移为基于NPU可运行的脚本。

迁移步骤

步骤1 通过以下任一方式启动脚本迁移任务。

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

步骤2 单击“Transfer”页签，进入迁移任务界面。

步骤3 参数配置。

“Transfer”界面如**图4-4**所示，用户自行根据实际情况配置参数。

图 4-4 Transfer 参数配置

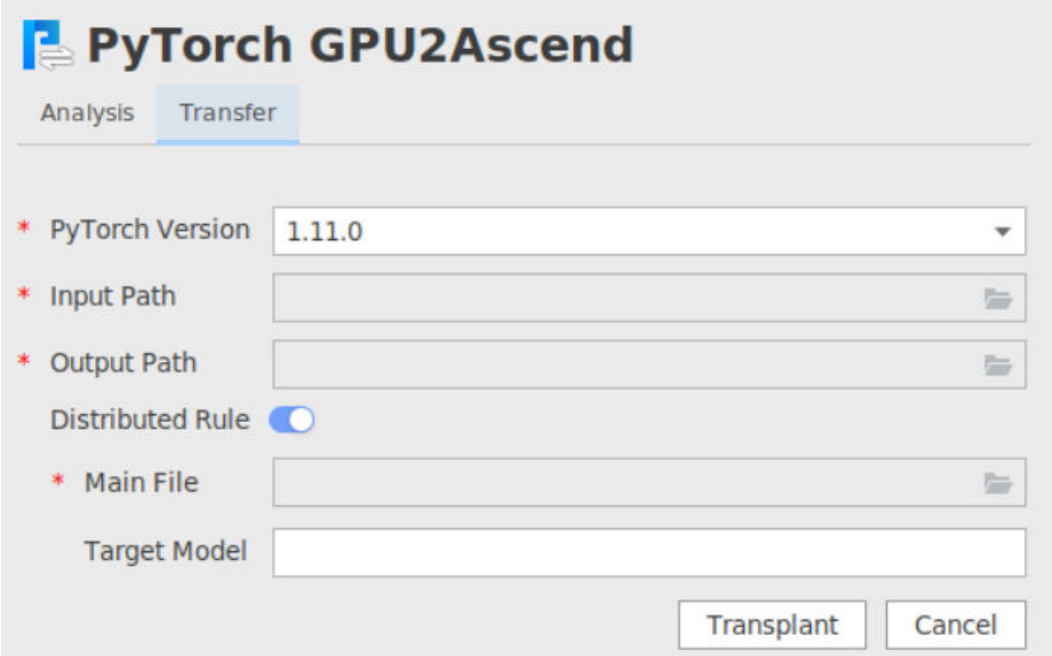


表 4-6 Transfer 参数配置

参数	参数说明
PyTorch Version	待迁移脚本的PyTorch版本。目前支持1.11.0、2.0.1、2.1.0。 必选，默认为1.11.0。
Input Path	待迁移脚本文件所在目录。单击文件夹图标选择目录。必选。
Output Path	脚本迁移结果文件输出路径。单击文件夹图标选择目录。必选。 • 不开启“Distributed Rule”即迁移至单卡脚本场景下，输出目录名为xxx_msft。 • 开启“Distributed Rule”即迁移至多卡脚本场景下，输出目录名为xxx_msft_multi。 xxx为原始脚本所在文件夹名称。

参数	参数说明
Distributed Rule	将GPU单卡脚本迁移为NPU多卡脚本，此参数**仅支持使用torch.utils.data.DataLoader方式加载数据的场景**。可选。 开启后，需配置如下参数： Main File：必选，单击文件夹图标选择训练脚本的入口Python文件。 Target Model：可选，待迁移脚本中的实例化模型变量名，默认为“model”。 如果变量名不为"model"时，则需要配置此参数，例如"my_model = Model()"，需要配置为**-t my_model**。

步骤4 单击“Transplant”，执行迁移任务。

完成后，**Output Path**输出目录下查看结果文件。

```
├── xxx_msft/xxx_msft_multi              // 脚本迁移结果输出目录
│   ├── 生成脚本文件                  // 与迁移前的脚本文件目录结构一致
│   ├── transplant_result_file       //存放迁移结果文件
│   │   ├── msFmkTranspltlog.txt         // 脚本迁移过程日志文件，日志文件限制大小为1M，若超过限制将分多个文件进行存储，最多不会超过10个
│   │   ├── cuda_op_list.csv            //分析出的cuda算子列表
│   │   ├── unknown_api.csv              //支持情况存疑的API列表
│   │   ├── unsupported_api.csv          //不支持的API列表
│   │   ├── api_precision_advice.csv     //API精度调优的专家建议
│   │   ├── api_performance_advice.csv  //API性能调优的专家建议
│   │   ├── change_list.csv              // 修改记录文件
│   ├── run_distributed_npu.sh       // 多卡启动shell脚本
```

----结束

后续操作

- 为了提升模型运行速度，建议开启使用二进制算子，参考《**CANN 软件安装指南**》安装二进制算子包后，参考如下方式开启：
  - 单卡场景下，修改训练入口文件例如main.py文件，在import torch_npu下方添加加粗字体信息。
    ```
    import torch
    import torch_npu
    torch.npu.set_compile(jit_compile=False)
    ......
    ```
  - 多卡场景下，如果拉起多卡训练的方式为mp.spawn，则torch.npu.set_compile(jit_compile=False)必须加在进程拉起的主函数中才能使能二进制，否则使能方式与单卡场景相同。
    ```
    if is_distributed:
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
    else:
        main_worker(args.gpu, ngpus_per_node, args)
    def main_worker(gpu, ngpus_per_node, args):
        # 加在进程拉起的主函数中
        torch.npu.set_compile(jit_compile=False)
        ......
    ```
- 如果启用了Distributed Rule参数，迁移后会生成如下run_distributed_npu.sh文件，在执行迁移后的模型之前需要把文件中的“please input your shell script

here”语句替换成模型原来的训练shell脚本。执行run_distributed_npu.sh文件后会生成指定NPU的log日志。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29688
export HCCL_WHITELIST_DISABLE=1

NPUS=($(seq 0 7))
export RANK_SIZE=${#NPUS[@]}
rank=0
for i in ${NPUS[@]}
do
    export DEVICE_ID=${i}
    export RANK_ID=${rank}
    echo run process ${rank}
    please input your shell script here > output_npu_${i}.log 2>&1 &
    let rank++
done
```

表 4-7 参数说明

参数	说明
MASTER_ADDR	指定训练服务器的IP
MASTER_PORT	指定训练服务器的端口号
HCCL_WHITELIST_DISABLE	hccl后端环境
NPUS	指定在特定NPU上运行
RANK_SIZE	指定调用卡的数量
DEVICE_ID	指定使用的device_id
RANK_ID	指定调用卡的逻辑ID

- 若用户训练脚本中包含昇腾NPU平台不支持的**amp_C**模块，需要用户手动删除后再进行训练。
- 若用户训练脚本中包含昇腾NPU平台不支持的**torch.nn.DataParallel**接口，需要手动修改成**torch.nn.parallel.DistributedDataParallel**接口执行多卡训练，参考**迁移单卡脚本为多卡脚本**进行修改。
- 若用户训练脚本中包含昇腾NPU平台不支持的**torch.cuda.default_generators**接口，需要手动修改为**torch_npu.npu.default_generators**接口。
- 若用户训练脚本中包含**torch.cuda.get_device_capability**接口，迁移后在昇腾AI处理器上运行时，会返回“None”值，如遇报错，需要用户将“None”值手动修改为固定值。**torch.cuda.get_device_properties**接口迁移后在昇腾AI处理器上运行时，返回值不包含minor和major属性，建议用户注释掉调用minor和major属性的代码。
- 分析迁移后可以参考**5 模型训练**进行训练。

4.3.5 使用 torch.utils.data.DataLoader 方式加载数据的场景说明

torch.utils.data.DataLoader是PyTorch中一个用于数据加载的工具类，主要用于将样本数据划分为多个小批次(batch)，以便进行训练、测试、验证等任务，查看模型脚本中的数据集加载方式是否是通过**torch.utils.data.DataLoader**加载，示例代码如下：

```
import torch
from torchvision import datasets, transforms
```

```
# 定义数据转换
transform = transforms.Compose([
    transforms.ToTensor(), # 将图像转换为张量
    transforms.Normalize((0.5,), (0.5,)) # 标准化图像
])
# 加载MNIST数据集
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
# 创建数据加载器
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=4)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=4)
# 使用数据加载器迭代样本
for images, labels in train_loader:
    # 训练模型的代码
    ...
```

4.3.6 FAQ

4.3.6.1 “Segmentation fault”错误

问题现象描述

运行转换后代码无报错，仅提示“Segmentation fault”信息。

原因分析

可能原因一：

代码中引用了tensorboard或第三方库中包含tensorboard，以下为已知的引用tensorboard的第三方库。

- wandb：若该库仅用来打log，可以删除该库的调用。
- transformers：该库深度绑定tensorflow、tensorboard。

可能原因二：

训练脚本中包含两个0维Tensor在不同设备上进行比较的代码，当前该比较不支持在torch_npu上运行。

解决措施

原因一解决措施：

注释掉相关的Summary、Writer调用即可规避该错误。Summary、Writer多用于记录日志和绘图，不影响网络跑通和精度收敛。

原因二解决措施：

在脚本启动命令前添加**python -X faulthandler**打印线程信息，定位到具体的报错位置，进行pdb调试。来定位脚本中是否存在两个0维Tensor在不同设备上进行比较的代码，需要用户手动修改为在同一设备上进行比较，示例如下：

修改前，在CPU和NPU上进行比较：

```
a = torch.tensor(123)
b = torch.tensor(456).npu()
print(a == b)
```

修改后，添加加粗字体信息，修改为同在NPU上进行比较：


```
a = torch.tensor(123).npu()
b = torch.tensor(456).npu()
print(a == b)
```

4.3.6.2 Muls 算子不支持 INT64

```
for epoch_num in iterator:
    for i_batch, sampled_batch in enumerate(trainloader):
        start_time = time.time()
        image_batch, label_batch = sampled_batch['image'], sampled_batch['label']
        image_batch, label_batch = image_batch.npu(), label_batch.npu()
        outputs = model(image_batch)
        loss_ce = ce_loss(outputs, label_batch[:].long())
        loss_dice = dice_loss(outputs, label_batch, softmax=True)
        loss = 0.5 * loss_ce + 0.5 * loss_dice
        optimizer.zero_grad()
```

将如上图中label_batch.npu()改成label_batch.int().npu()，即把当前报错行的变量类型改成int32，规避此类Muls算子不支持int64的问题。

4.3.6.3 对于报错 “No supported Ops kernel and engine are found for [ReduceStdV2A], optype [ReduceStdV2A]”，算子 ReduceStdV2A 不支持的问题

可以通过用std求标准差再平方得到var，均值单独调用mean接口求来规避问题例如：

```
>>> a = torch.randn(2,2,4,4).npu()
>>> v = torch.std(a)**2
>>> m = torch.mean(a)
>>> v
tensor(0.9687, device='npu:0')
>>> m
tensor(-0.0429, device='npu:0')
```

具体到代码中修改：

```
class StdConv2d(nn.Conv2d):

  def forward(self, x):
    w = self.weight
    # v, m = torch.var_mean(w, dim=[1, 2, 3], keepdim=True, unbiased=False)
    v = torch.std(w) ** 2
    m = torch.mean(w)
    w = (w - m) / torch.sqrt(v + 1e-10)
    return F.conv2d(x, w, self.bias, self.stride, self.padding,
                    self.dilation, self.groups)
```


4.3.6.4 运行报错

表 4-8 运行报错

报错信息	解决措施
运行报错：RuntimeError: Attempting to deserialize object on a CUDA device but torch.cuda.is_available() is False. If you are running on a CPU-only machine, please use torch.load with map_location=torch.device（‘cpu’) to map your storages to the CPU.	一般在报错代码行加入参数map_location=torch.device（‘cpu’）即可规避此问题。
运行报错：Unsupport data type: at::ScalarType::Double.	在报错代码行前添加数据类型转换语句即可规避此问题。 如报错代码行为pos = label.data.eq(1).nonzero(as_tuple =False).squeeze().npu()不支持数据类型，在代码上一行加上label = label.cpu().float().npu()进行数据类型转换。

运行报错：IndexError: invalid index of a 0-dim tensor. Use tensor.item() in Python or tensor.item<T>() in C++ to convert a 0-dim tensor to a number.	遇到类似错误直接将代码中.data[0]改成.item().
	例如将：
	`M = (d_loss_real + torch.abs(diff)).data[0]`
	改为：
	`M = (d_loss_real + torch.abs(diff)).item()`

运行报错：Could not run ‘aten::empty_with_format’ with arguments from the ‘CPUTensorId’ backend. ‘aten::empty_with_format’ is only available for these backend “CUDA、NPU.	需要将tensor放到npu上，类似input = input.npu()。
运行报错：options.device().type() == DeviceType::NPU INTERNAL ASSERT FAILED xxx:	需要将tensor放到npu上，类似input = input.npu()。
运行报错：Attempting to deserialize object on a CUDA device but torch.cuda.is_availabel() is False.	此错误一般是torch.load()接口导致的，需要加关键字参数map_location，如map_location='npu’或map_location=‘cpu’。

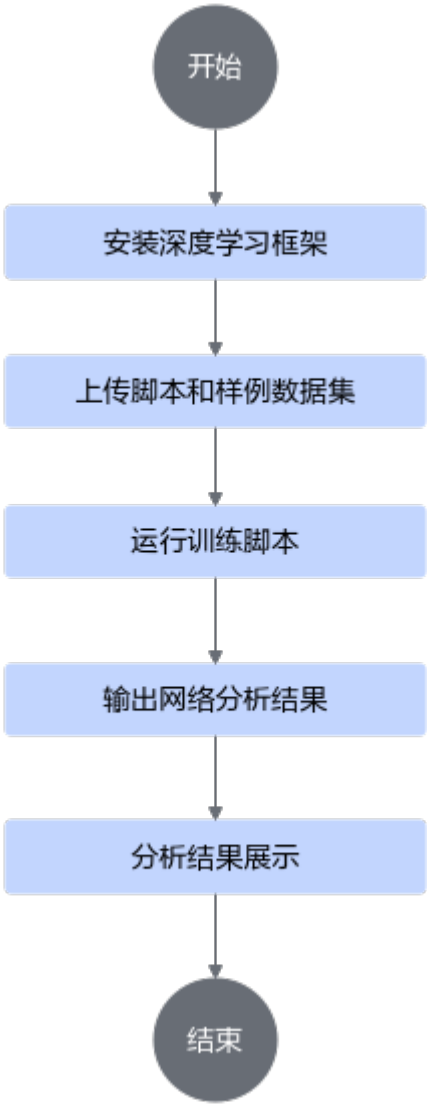
报错信息	解决措施
运行报错：RuntimeError: Incoming model is an instance of torch.nn.parallel.DistributedDataParallel. Parallel wrappers should only be applied to the model(s) AFTER.	此错误是由于“torch.nn.parallel.DistributedDataParallel”接口在“apex.amp.initial”接口之前调用导致的，需要手动将“torch.nn.parallel.DistributedDataParallel”接口的调用移到“apex.amp.initial”接口调用之后即可。

5 模型训练

5.1 简介

MindStudio仅支持MindSpore、TensorFlow和PyTorch框架的模型训练，把执行的脚本、数据集、参数等相关信息通过网络分析并输出分析结果。基于MindStudio的训练流程如**图5-1**所示。

图 5-1 业务流程



5.2 操作步骤

MindStudio执行模型训练的主要步骤包括：

1. **创建训练工程**（首次执行训练）/**导入训练工程**（已存在训练工程）
2. **运行配置**
3. **执行训练**

前提条件

- 当工程选择为MindSpore工程或样例时，需参见MindSpore官方网站**安装**页面进行运行环境的搭建。
- 当工程选择为TensorFlow工程或样例时，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“安装开发环境 > 在昇腾设备上安装 > 安装框架插件包”章节。

- 当工程选择为PyTorch工程或样例时，需安装PyTorch框架及混合精度模块，安装方式请参见《PyTorch网络迁移和训练指南》的“环境准备”章节。

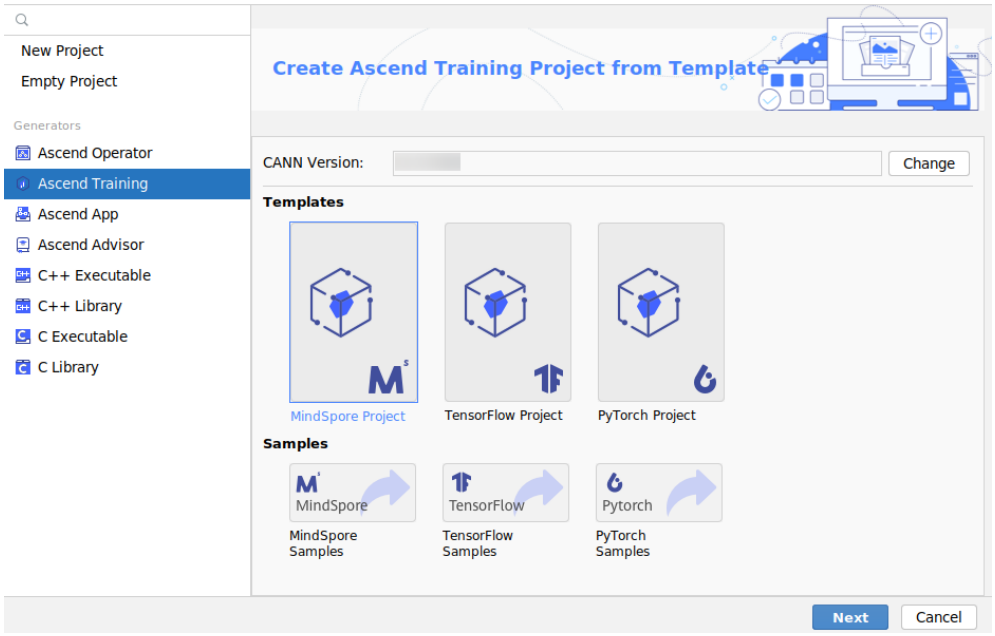
创建训练工程

当前支持以MindSpore、TensorFlow和PyTorch训练框架为模板创建训练工程，其中MindSpore训练框架可以使用MindSpore Insight实现训练过程可视化，MindSpore Insight详细使用方法请参见MindSpore Insight官方文档。

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。

- MindStudio欢迎界面：单击“New Project”，进入创建工程界面。
- MindStudio工程界面：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。

- 当选择“Templates”下的工程时
  a. 在左侧导航栏选择“Ascend Training”，如图5-2所示。
     i. 在右侧界面选择“CANN Version”。
     ii. 选择“Templates”下的工程，例如选择“MindSpore Project”。

     说明

     当选择“Templates”下的工程时，需要用户自行准备训练脚本：

     - 若为NPU训练脚本，可直接进行模型训练。
     - 若为GPU训练脚本，需要将脚本通过4 分析迁移转换为NPU训练脚本，再进行模型训练。

  b. 单击“Next”，配置训练工程其他信息，参数说明如表5-1所示。

表 5-1 工程参数说明

参数	说明
Project name	工程名称，自行配置。 ■ 名称开头和结尾必须是数字或字母。 ■ 只能包含字母、数字、中划线和下划线。 ■ 长度不超过64个字符。
Project location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）
More settings	“Module name”：模块名，默认与“Project name”一致。
	“Content root”：根目录下路径。
	“Module file location”：模块文件路径。
	单击“Project format”右侧选框，出现下拉菜单。 ■ .idea（directory-based）：创建项目的时候创建一个.idea的项目录来保存项目的信息，默认选项。 ■ .ipr（file-based）：项目配置文件来保存项目的配置信息。

c. 单击“Create”完成训练工程的创建。
若工作窗口已打开其他工程，会出现确认提示。
   - 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。

d. 查看训练工程目录结构和主要文件，请以实际创建结果为准。

```
├── .idea
├── data                          //数据集目录，需用自行创建
├── .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CANN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
```

- 当选择“Samples”下的工程时
  a. 在左侧导航栏选择“Ascend Training”，如图5-2所示。
     i. 在右侧界面选择“CANN Version”。
     ii. 选择“Samples”下的工程，例如选择“MindSpore”。
  b. 单击“Next”，跳转至gitee代码仓界面。
  c. 在gitee代码仓页面下单击“克隆/下载 > 复制”，复制代码包下载链接。
  d. 在开发环境执行命令：git clone URL（其中URL为复制的代码包下载链接），直接将代码包克隆到开发环境。

```
git clone https://gitee.com/mindspore/models.git
```


说明

TensorFlow和Pytorch代码包下载链接如下所示：

- TensorFlow：https://gitee.com/ascend/ModelZoo-TensorFlow
- Pytorch：https://gitee.com/ascend/ModelZoo-PyTorch

e. 在下载的文件夹中选择需要的模型样例，然后直接通过MindStudio导入，详情请参见**导入训练工程**。

说明

当通过Samples下载gitee代码仓中的模型样例时，可直接进行模型训练。

----结束

导入训练工程

如果已有训练工程，则无需新建训练工程，可直接通过MindStudio导入，操作如下。

步骤1 使用MindStudio导入训练工程。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的，选择现有工程打开。

说明

- 如该工程存在代码风险，在打开时会弹出信任窗口。
  - 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
  - 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
  - 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。
- 若导入NPU训练工程，可直接进行模型训练；若导入GPU训练工程，需要将脚本通过**4 分析迁移**转换为NPU训练脚本，再进行模型训练。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤3 成功导入工程后，工程目录以树状呈现，请以实际创建结果为准。

说明

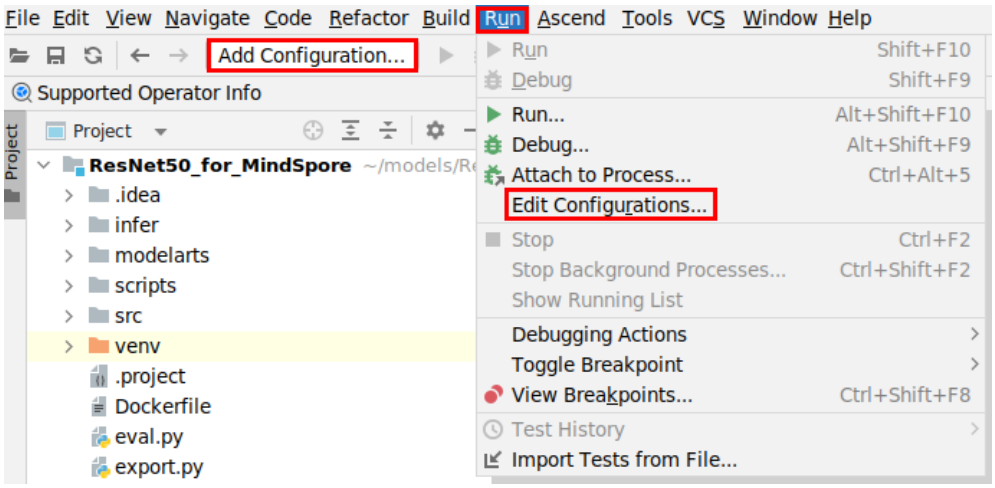
如果导入的是非昇腾工程，需要将工程转换为昇腾工程，再进行运行配置，详情请参见**2.1.3 工程转换**。

----结束

运行配置

步骤1 单击工程界面“Run > Edit Configurations...”或单击图5-3所示菜单中的“Edit Configurations...”，进入运行配置界面。

图 5-3 快捷方式进入运行配置界面



步骤2 配置训练参数。

说明

运行配置支持Ascend Training和Python两种配置方式（推荐使用Python配置方式）：

- Python：支持运行、调试工程以及训练过程中代码异常跳转功能。
- Ascend Training：仅支持运行工程。

- 使用Python配置方式

  如果训练工程中的执行源文件为*xxx*.py格式，推荐使用此配置方式。

  a. 单击左上角的“+”，在弹出的下拉框选择“Python”，新增文件的编译调试配置。

  b. 右侧显示配置项，配置示例如图5-4所示，关键配置参数参见表5-2。

图 5-4 源文件配置项

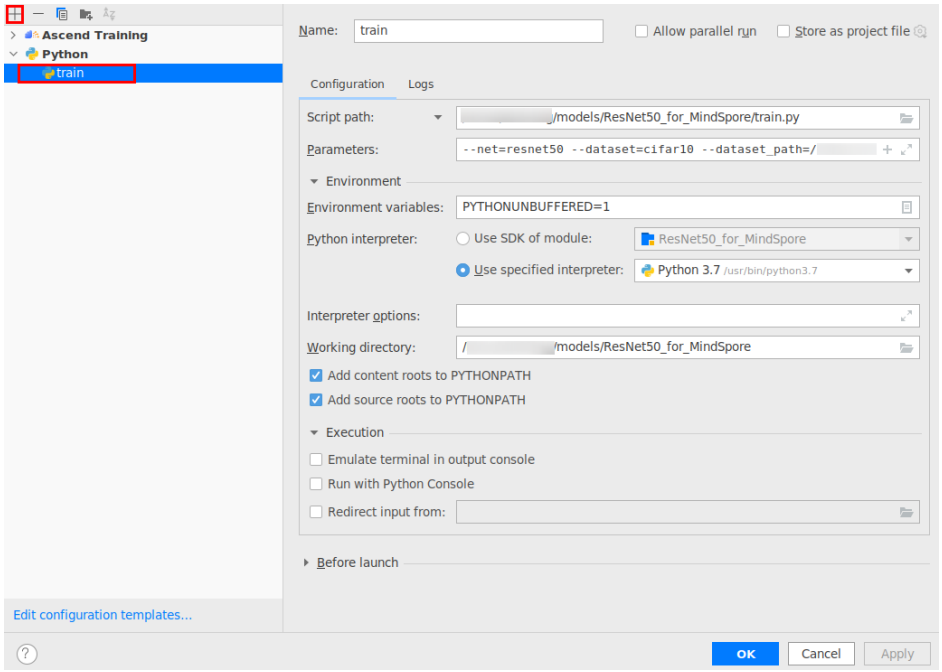


表 5-2 调试运行关键参数说明

参数	说明
Name	用户自行定义。
Script path	选择需要执行调试的Python源文件具体路径。
Parameters	运行参数，请根据实际情况自行配置。
Python interpreter	“Use SDK of module”：使用模块级的Python SDK进行解析。 具体配置功能请参见**设置模块级Python SDK（昇腾工程）**或**设置模块级Python SDK（非昇腾工程）**。
	“Use specified interpreter”：使用已配置的特定解析器。 具体配置功能请参见**13.9 Python SDK设置**。
Working directory	工作目录（默认为需执行调试的Python源文件所在的目录）。

c. 单击“Apply”应用后，单击“OK”保存并关闭调试配置界面。

- 使用Ascend Training配置方式

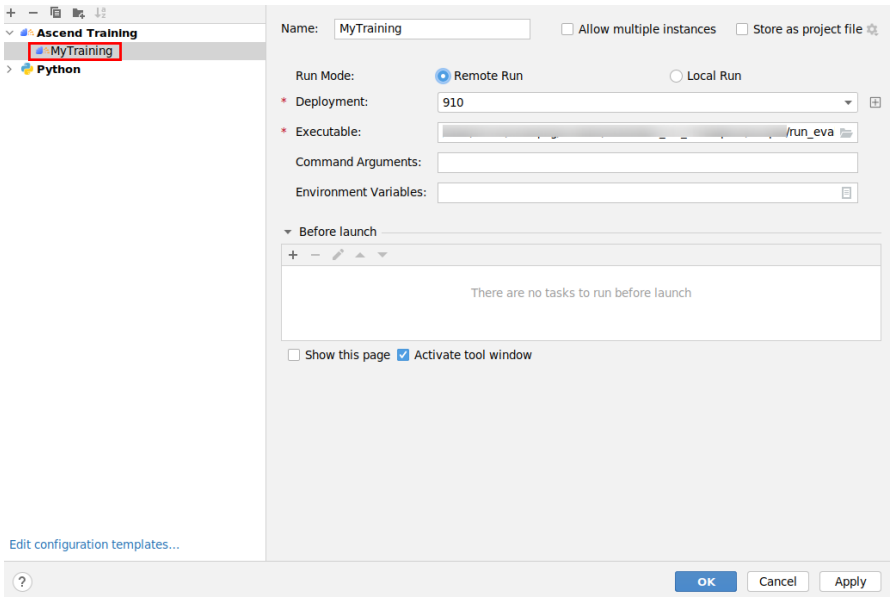
  如果训练工程中的执行源文件为*xxx*.sh格式，请使用此配置方式。

  a. 单击左上角的“+”，在弹出的下拉框选择“Ascend Training”，新增文件的运行配置。

  b. 右侧显示配置项，配置示例如**图5-5**和**图5 运行配置界面（Local Run）**所示，关键配置参数参见**表5-3**。

- 当“Run Mode”选择“Remote Run”时：

图 5-5 运行配置界面（Remote Run）



- 当“Run Mode”选择“Local Run”时：

图 5-6 运行配置界面（Local Run）

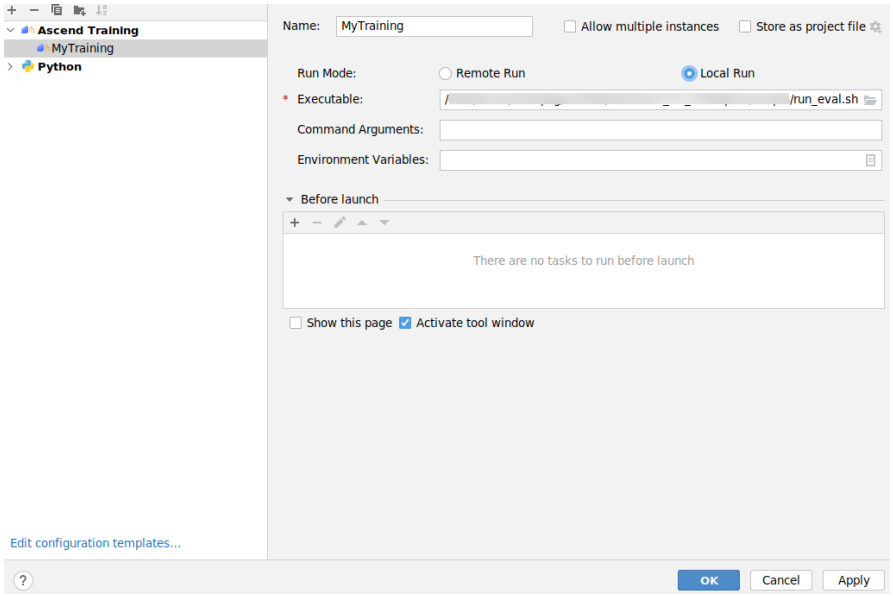


表 5-3 训练工程运行信息

参数	说明
Name	工程名称，用户自行配置，必选。 ■ 名称必须以字母开头，数字或字母结尾。 ■ 只能包含字母、数字、中划线和下划线。 ■ 长度不能超过64个字符。
Run Mode	运行环境选择。选择“Remote Run”或“Local Run”，默认为“Remote Run”。
Executable	训练工程中的执行源文件，必选。 例如：*$home*/AscendProjects/MyTraining/*xxx*。
Deployment	运行配置。选择Remote Run模式时可见，必选。 请参见13.3 Ascend Deployment进行配置，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
Command Arguments	训练工程执行参数，可选。
Environment Variables	训练工程环境变量，可选。

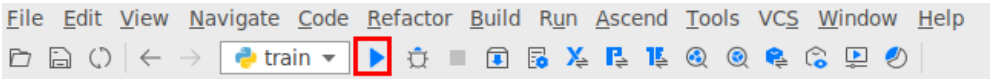
c. 单击“Apply”应用后，单击“OK”保存并关闭运行配置界面。

----结束

执行训练

步骤1 单击工程界面“Run > Run 'train'”或单击图5-7所示菜单，执行训练。

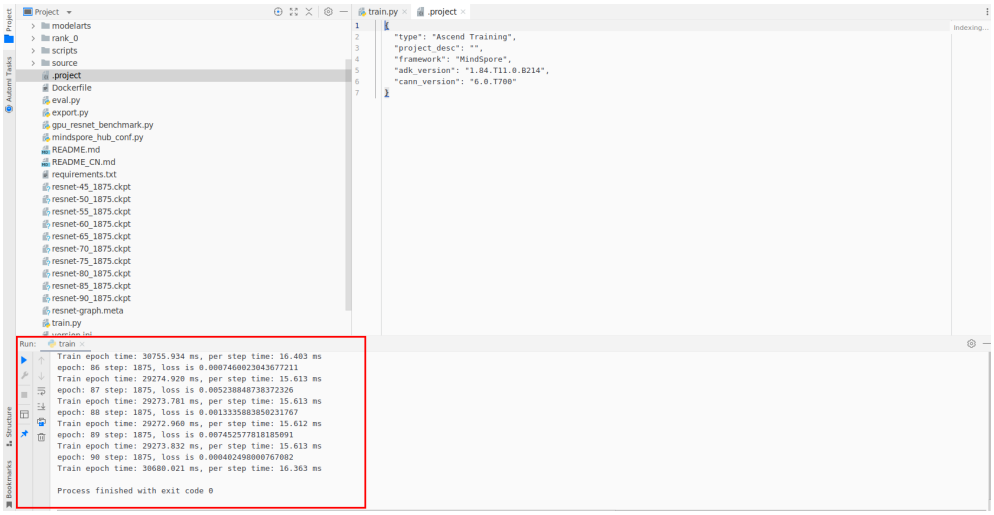
图 5-7 快捷方式执行训练



步骤2 查看训练结果。

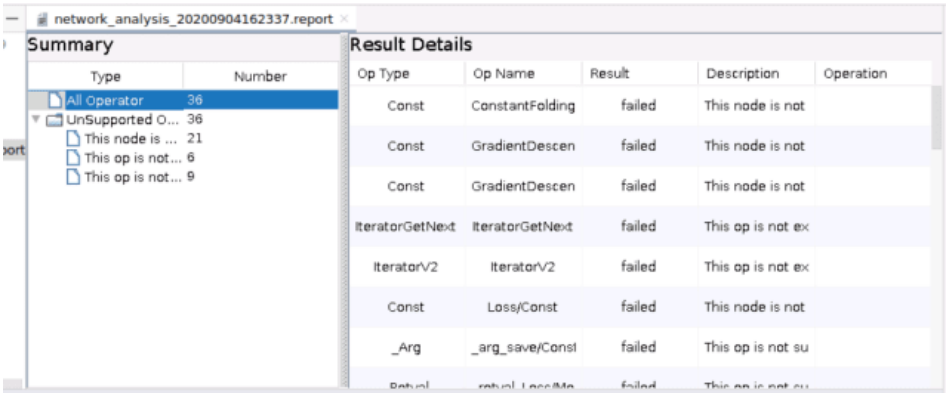
- 训练成功：在工程界面底部“Run”窗口显示运行实时信息，如图5-8所示。

图 5-8 运行实时信息



- 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*。报告内容如图5-9所示。

图 5-9 整网支持度报告



说明

图7 整网支持度报告是以MindSpore训练框架创建的训练工程，训练失败后在工程根目录下的“out/reports”下生成的*.report*文件；以TensorFlow和PyTorch训练框架创建的训练工程，训练失败后生成的*.report*文件返回至用户自定义的输出路径。

----结束

6 模型转换和调优

6.1 简介

模型转换和调优是将Caffe/TensorFlow等框架训练好的模型，通过ATC或者AOE工具将其转换为昇腾AI处理器支持的离线模型，详细架构如**图6-1**和**图6-2**所示。

- ATC工具：ATC会进行算子调度优化、权重数据重排、内存使用优化等具体操作，对原始的深度学习模型进行进一步的调优，从而满足部署场景下的高性能需求，使其能够高效执行在昇腾AI处理器上。
- AOE工具：AOE通过生成调优策略、编译、在运行环境上验证的闭环反馈机制，不断迭代出更优的调优策略，最终得到最佳的调优策略，从而可以更充分利用硬件资源，不断提升网络的性能，达到最优的效果。两种调优方式如下：
  - 子图调优：通过SGAT（SubGraph Auto Tune），对子图切分策略进行调优，通过在运行环境上验证获得真实性能，最终将最优的调优策略固化至模型知识库，并获取优化后的模型。
  - 算子调优：通过OPAT（Operator Auto Tune），对算子进行调优，通过在运行环境上验证获取真实性能，最终将优选算子调优策略固化到算子知识库。

图 6-1 ATC 工具功能架构

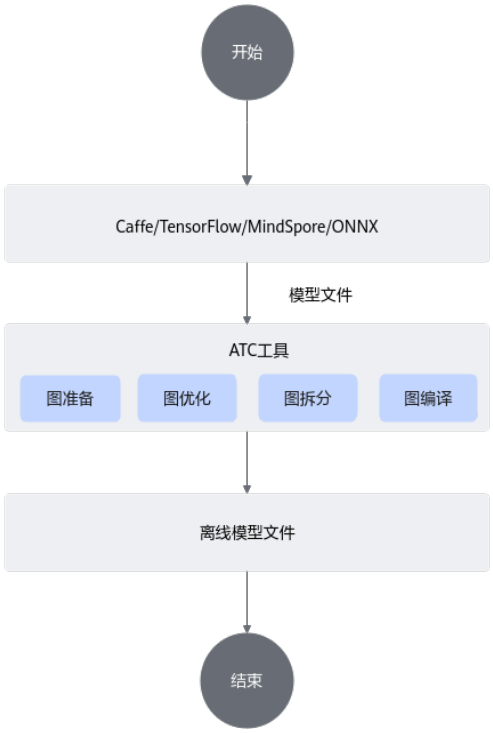
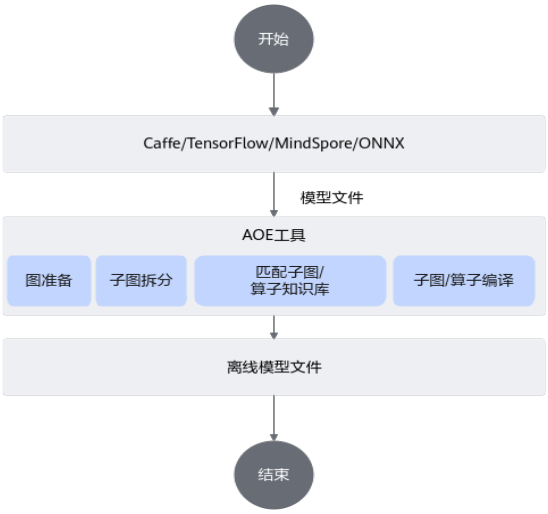


图 6-2 AOE 工具功能架构



6.2 使用约束

- 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， 请参见《**ATC工具使用指南**》“定制网络专题”章节先修改prototxt模型文件。
- 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。模型转换和调优时需指定固定数值。
- 模型中的所有层算子除常量算子外，输入和输出需要满足维度不为0。

- 只支持《CANN 支持Caffe&TensorFlow&ONNX算子清单》中的算子，并需满足算子限制条件。
- 模型调优的使用约束请参见《AOE工具使用指南》中的“离线推理场景下调优（其他推理设备）> 执行调优”章节。

6.3 支持框架类型

模型转换和调优支持的原始框架类型、输入数据类型和模型文件如表6-1所示。

表 6-1 支持框架类型

原始框架类型	输入数据类型	模型文件说明
Caffe	• FP32 • FP16：通过设置入参--input_fp16_nodes实现。 • UINT8：通过配置数据预处理实现。 **说明** 输入数据最大支持四维，转维算子（reshape、expanddim等）不能输出五维。	• 模型文件：*xxx*.prototxt • 权重文件：*xxx*.caffemodel 其中，模型文件和权重文件的op_name、op_type必须保持名称一致（包括大小写）。
TensorFlow	• FP16 • FP32 • UINT8 • INT32 • INT64 • BOOL **说明** 不支持输出数据类型为INT64，需要用户自行将INT64的数据类型修改为INT32类型。	模型文件：*xxx*.pb 只支持FrozenGraphDef格式的.pb模型转换和调优。
ONNX	• FP32 • FP16：通过设置入参--input_fp16_nodes实现。 • UINT8：通过配置数据预处理实现。	模型文件：*xxx*.onnx
MindSpore	• FP32 • UINT8：通过配置数据预处理实现。	模型文件：*xxx*.air

6.4 执行转换

模型转换和调优入口

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

  转换详情请参见**操作步骤**。
- 用户也可以通过如下方式使用ATC或者AOE工具进行模型转换和调优。
  a. 单击MindStudio界面下方的Terminal窗口。
  b. 参见《**ATC工具使用指南**》或者《**AOE工具使用指南**》使用命令行方式进行模型转换和调优。

前提条件

- 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Ascend-cann-toolkit开发套件包所在的开发环境。
- 使用Atlas 200/500 A2推理产品进行AOE调优时，避免由于环境配置太低或者内存不足的情况导致AOE调优失败，建议开启NCS服务，在开发环境和运行环境之间建立通信，具体操作请参见《**AOE工具使用指南**》中“配置密钥证书”章节。

  说明

  开启NCS服务后，进行AOE调优时需在“Advanced Options Preview”高级选项配置页签的“Additional Arguments”参数中配置“--ip=xx.xx.xx.xx --port=xxx”与运行环境建立通信，IP和Port为运行环境的IP和端口号。

操作步骤

以下操作以ATC工具为例：

步骤1 打开模型转换和调优页面，在“Model Information”页签中上传模型文件和权重文件，界面参考如**图6-3**或**图6-4**所示。

图 6-3 配置模型信息（Linux 系统示例）

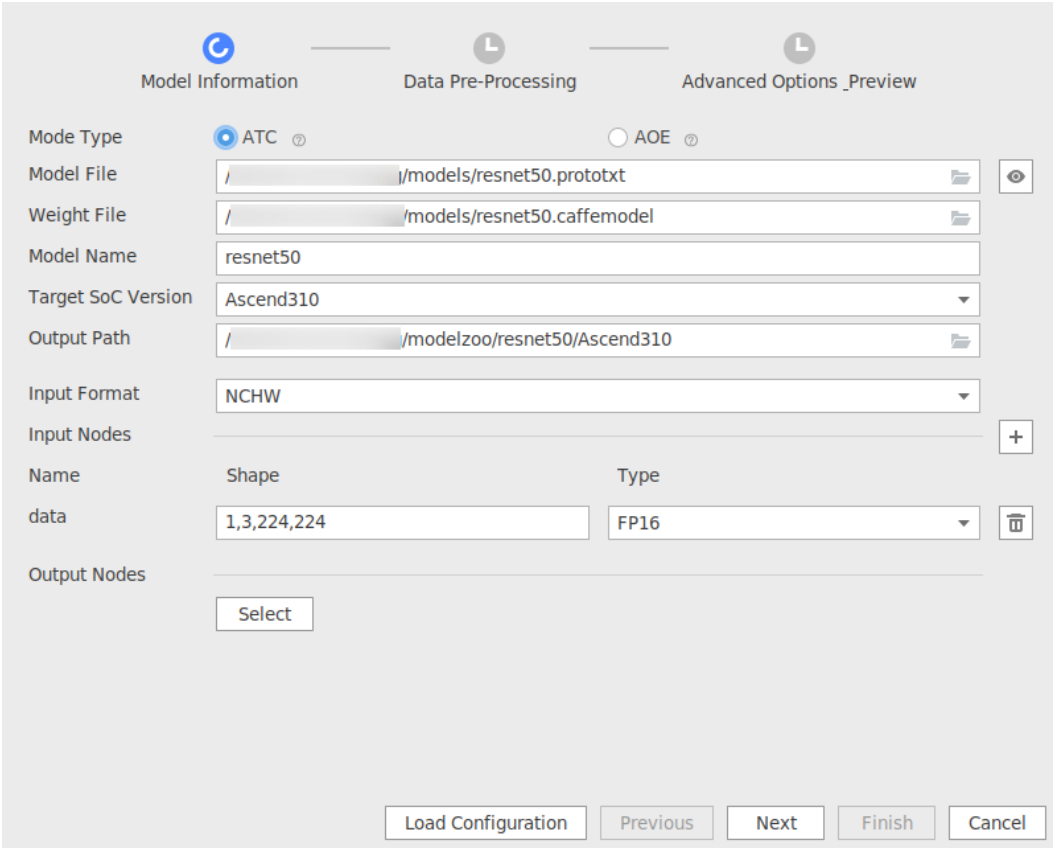
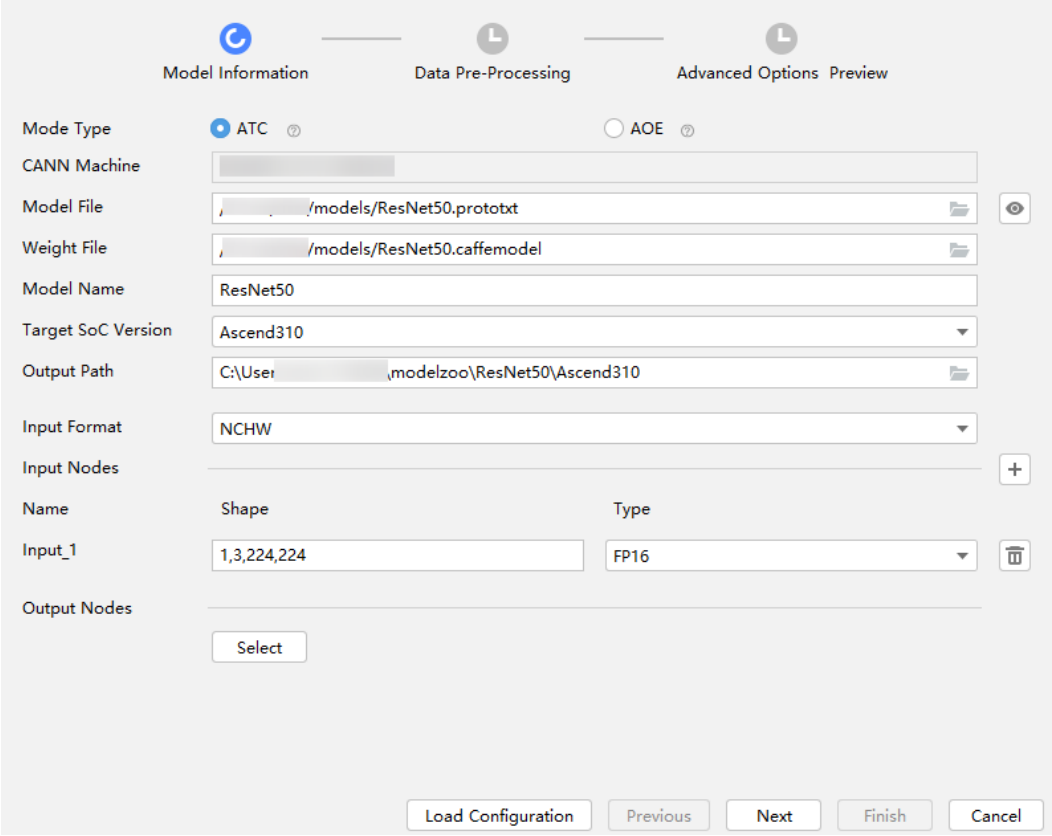


图 6-4 配置模型信息（Windows 系统示例）



参数解释如表6-2所示。

表 6-2 Model Information 界面参数配置

参数	说明	备注
Mode Type	运行模式，默认ATC。 必选。	• ATC：将开源框架的网络模型转换为适配昇腾AI处理器的离线模型，此模式不会对模型进行调优。 • AOE：充分利用有限的硬件资源，对模型进行算子调优或者子网调优，以满足算子和整网的性能要求。
CANN Machine	远程连接CANN所在环境的SSH地址。 必选。	仅Windows系统支持此参数，表现格式为*<username>@localhost:端口号*。

参数	说明	备注
Model File	模型文件，该模型文件需要取消其他用户写的权限。 必选。	• Linux环境：单击右侧的或直接输入本地服务器上需要转换的模型文件路径。 • Windows环境： – 选择“remote path”，在后台服务器路径选择需要转化的模型文件并上传； – 选择“local path”，单击右侧的，在Windows本地选择或直接输入需要转化的模型文件路径并上传。 选择好模型文件后，单击右侧的按钮，可以查看该模型的原始网络结构图，详情请参见**•Model File：查看原始模型网络结构图**。 **说明** • 当导入超大模型时，如果提示报错“Failed to get model input shape.”，请在菜单栏选择“Help > Change Memory Settings”，在弹出的Memory Settings窗口中增大内存。 • 配置界面只支持对单个模型进行AOE调优，如需对目录下多个模型进行调优，请在MindStudio界面下方的Terminal窗口执行AOE调优命令。例如：aoe --job_type=1 --model_path=*$home/xxx/*。*$home/xxx/*为待调优模型所在路径，用户根据实际路径自行替换。
Weight File	权重文件。 当原始框架是Caffe时，该参数存在且必选。	• Windows环境中： 模型文件和权重文件需要放在同一目录下，单击选择模型文件时，权重文件将会自动填充。 • Linux环境中： – 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或者在同一目录下，但名称和模型文件名称不相同。单击右侧的选择模型文件对应的权重文件或在输入框中自行输入*.caffemodel权重文件在后台服务器的路径。
Model Name	输出的模型文件名称。 必选。	• 选择模型文件后，该参数会自动填充，用户可以自行修改。 • 如果模型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。
Job Type	调优模式。 Mode Type选择AOE时必选。	取值如下所示，默认为1-Subgraph Auto Tune(SGAT)。 • 1-Subgraph Auto Tune(SGAT)：表示子图调优。 • 2-Operator Auto Tune(OPAT)：表示算子调优。

参数	说明	备注
Target SoC Version	模型转换时指定芯片型号。 Mode Type选择ATC时必选。	请根据板端环境具体芯片形态进行选择。
Output Path	模型文件输出路径。 必选	默认输出路径为$HOME/modelzoo/${Model Name}/${Target SoC Version}/，也可手动输入或单击可单击右侧 进行自定义。
Input Format	输入数据格式。 必选。	该参数只支持设置为单个取值。 • 当原始框架是Caffe时，取值为NCHW、ND（表示支持任意维度格式，N<=4），默认为NCHW。 • 当原始框架是ONNX时，取值为NCHW、NCDHW、ND（表示支持任意维度格式，N<=4），默认为NCHW。 • 当原始框架是MindSpore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。
Input Nodes	模型输入节点信息。 必选。	• 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Type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信息和输入节点的数据类型Type。 • 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信息和Type信息。 **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。 – 模型转换工具目前不支持删除模型中默认带动态Shape的节点。
Shape	模型输入的shape信息。	例如图6-3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。 更多输入数据格式设置请参见**•Shape：根据输入数据格式分为以下两种设置情...**。

参数	说明	备注
Type	指定输入节点的数据类型。 必选。	如果模型有多个输入，只有Type取值不为FP16的节点，才可以配置“Data Pre-Processing”页签；如果Type取值不为FP16的节点无法获取Shape中的H，W信息，也无法配置“Data Pre-Processing”页签。 更多原始框架支持的数据类型以及是否可配置“Data Pre-Processing”页签情况请参见**•Type：支持的数据类型**。
Output Nodes	指定输出节点信息。 可选。	如果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，详细操作请参见**•Output Nodes：指定输出节点信息**。模型转换后，在相应.om模型文件可以看到该层算子的输出直接作为模型的输出。详细信息请参见**6.5 模型可视化**。 “Output Nodes”参数下方“Select”层的算子，默认为全部选中。
Load Configuration	导入上次模型转换的配置文件。 可选。	如果用户之前转换过模型，无论成功与否，在$HOME/modelzoo/${Model Name}/${Target SoC Version}/路径都会生成${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。

Model Information界面参数详细说明：

- Model File：查看原始模型网络结构图。

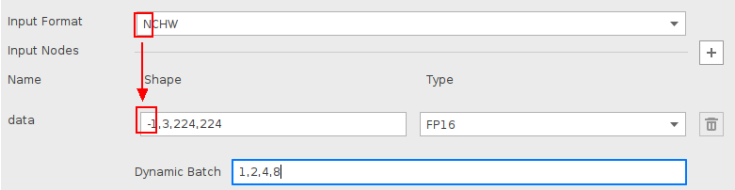
  在“Model File”栏选择好模型文件后，单击右侧的  按钮，弹出生成模型网络结构图进度条，之后会弹出该模型的原始网络结构图，在模型网络结构图中可以进行以下操作，具体操作方法请参见**可视化界面说明**（MindSpore框架的原始网络模型，不支持查看网络结构图。）。

  - **查看算子信息**
  - **查看算子输出维度和shape信息**

    如果用户选择的模型中包括不支持的算子，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**
  - **搜索算子内部信息**
- Shape：根据输入数据格式分为以下两种设置情况。
  - 当“Input Format”参数是具有固定形状的输入数据格式，如NCHW、NCDHW等。

- 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。

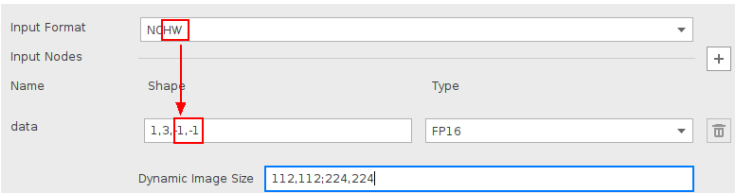
  将解析的shape中的N设置为-1，会在shape下方出现**Dynamic Batch Size**参数。在其中的编辑框中输入具体的档位数，每一档通过英文逗号分隔。最多支持100档配置，每个档位数值建议限制为：[1~2048]，例如输入1,2,4,8。

  如果模型转换时设置了**Dynamic Batch 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**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《**AscendCL应用软件开发指南（C&C++）**》中的“AscendCL API参考 > 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，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，最少输入两组，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。

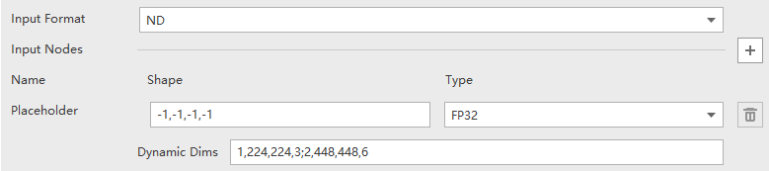
  如果模型转换时设置了**Dynamic Image Size**参数，则使用应用工程进行模型推理时，需要在**aclmdlExecute**接口之前，增加**aclmdlSetDynamicHWSize**接口，用于设置真实的分辨率。关于**aclmdlSetDynamicHWSize**接口的具体使用方法，请参见《**AscendCL应用软件开发指南（C&C++）**》中的“AscendCL API参考 > 模型加载与执行”章节。

  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。

  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。

- 当“Input Format”参数为ND，设置ND格式下的动态维度：适用于执行推理时，每次处理任意维度的场景。

  根据需要设置-1在Shape中数量及位置，会在**Shape**下方出现**Dynamic Dims**参数。在其中的编辑框中输入具体的动态维度参数，最少输入2组，最多100组。组与组之间使用英文分号分隔，组内参数使用英文逗号分隔。组与组之间的内容不能重复，组内的最小维度为1。每组中的参数值与**Shape**参数中的-1标识的参数依次对应，**Shape**参数中有几个-1，则每组必须设置几个参数值。例如输入shape信息为“-1,-1,-1,-1”，输入**Dynamic Dims**参数可以为“1,224,224,3;2,448,448,6”。



- Type：支持的数据类型。
  - 支持的数据类型：
    - 若原始框架类型为Caffe、ONNX，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TensorFlow，支持的输入数据类型为FP32、FP16、UINT8、Int32、Int64、Bool。
  - 根据Type数据类型判断“Data Pre-Processing”页签是否可配置：
    - 当原始框架为Caffe、ONNX和MindSpore时，只有Type取值为UINT8，**步骤2**中“Data Pre-Processing”页签才支持配置，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。
    - 当原始框架为TensorFlow时，只有Type取值不为FP16，**步骤2**中“Data Pre-Processing”页签才支持配置：
      - 如果Type取值为UINT8，则“Data Pre-Processing”页签默认开启，且不可以关闭。
      - 如果Type取值为FP32、Int32、Int64、Bool中的一种，则“Data Pre-Processing”页签默认关闭，可手动开启。
- Output Nodes：指定输出节点信息。

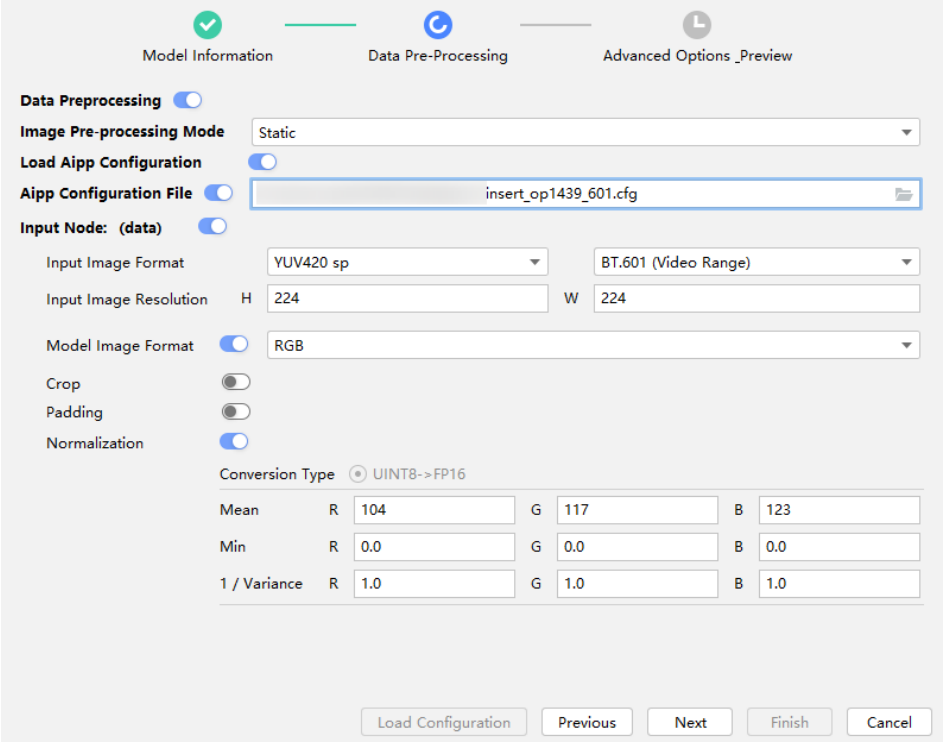
  单击“Select”在弹出的网络拓扑结构中，选中某层节点，右击选择“Select”，该层由蓝色标签进行标记，单击“OK”后，在“Output Nodes”参数下面会看到标记层的算子，右击选择“Deselect”取消选中。
  - Op Name：标记层的算子名称。
  - Data Type：算子输出的数据类型，包括FP32、UINT8、FP16，通过该参数用户可以设置单个算子的输出数据类型。

  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。

步骤2 单击“Next”，进入“Data Pre-Processing”配置数据预处理页签，界面参考如**图6-5**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。

图 6-5 配置数据预处理功能



参数说明如表6-3所示。

表 6-3 Data Pre-Processing 界面参数配置

参数	说明	备注
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	备注
Load Aipp Configuration	AIPP配置加载功能。动态AIPP不支持此功能。	此功能默认关闭，开启此功能后，在“Aipp Configuration File”输入框中选择对应的配置文件。 加载配置文件后，所有“Input Node: (data)”和“Input Node: (*im_info*)”下的参数按照配置文件自动设置。如有需要，打开“Aipp Configuration File”右侧的开关，界面下方将会展示“Input Node: (data)”和“Input Node: (*im_info*)”中的所有参数，用户可对其进行自定义修改。
Input Node: (data)	节点配置开关，可以控制是否对该节点开启AIPP。	• 当**图6-3**中Input Nodes参数对应data节点的Type取值为UINT8，并且能获取模型的宽和高，该参数才会自动开启。 • 当模型为TensorFlow时，**图6-3**中Input Nodes参数对应data节点的Type取值为FP32、Int32、Int64、Bool，并且能获取模型的宽和高时，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6-3**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，该参数才会自动开启。Input Node: (*im_info*)中的*im_info*根据解析的模型不同而变化。
Input Image Format	输入图片格式。	更多图片输入格式详情请参见**•Input Image Format：根据图片...**。
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。 • 如果**1**中“Shape”设置了动态分辨率参数，即原始图片的宽和高不确定，要求其宽和高取值为0。
Model Image Format	模型处理图片格式。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。	输入图片格式不同，模型处理图片格式不同，更多格式说明请参见**•Model Image Format：根据图片...**。
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，参数说明以及抠图约束请参见**•Crop：抠图开关开启后，参数解释以及抠图约束...**。
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。

参数	说明	备注
Normalization	归一化开关。	开启后，其中“Conversion Type”表示计算规则，包括Mean、Min、Variance三个配置项。
Mean	每个通道的均值。	当开启“Normalization”参数时才呈现，该参数的具体显示以及值的详情请参见**•Mean：根据“Model Image For...**。
Min	每个通道的最小值。	当开启“Normalization”参数时才呈现，该参数的具体显示以及值的详情请参见**•Min：根据“Input Image Form...**。
1/Variance	每个通道的方差的倒数。	当开启“Normalization”参数时才呈现，该参数的具体显示以及值的详情请参见**•1/Variance：根据“Input Ima...**。
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。	• 当**图6-3**中Input Nodes参数对应data节点的Type取值为UINT8，并且能获取模型的宽和高，该参数才会自动开启。 • 当模型为TensorFlow时，**图6-3**中Input Nodes参数对应data节点的Type取值为FP32、Int32、Int64、Bool，并且能获取模型的宽和高时，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动态AIPP。	只有**图6-3**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，该参数才会自动开启。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）。	根据输入图像格式确定该参数的取值，详情请参见**•Max Image Size (Byte)：根...**。

Data Pre-Processing界面参数详细说明：

- Input Image Format：根据图片输入格式，分为以下几种情况。
  - 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：

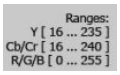
    BT.601(Video Range)、BT.601(Full Range)、BT.709(Video Range)、BT.709(Full Range)

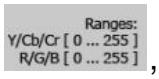
    不同的类型，对应不同的色域转换配置（色域转换，用于将输入的数据格式，转换为模型需要的格式，具体色域转换系数会在模型转换完成后生成在insert_op.cfg配置文件中）其中：

    - 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： 

；WIDE取值范围为： 

，关于如何判断输入数据的标准，请参见《**ATC工具使用指南**》“FAQ > 使用AIPP色域转换模型时如何判断视频流的格式标准”章节。
- 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：
    ```
    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
    ```

- Model Image Format：根据图片格式输入不同，模型处理图片格式不同，分为以下几种情况。
  - 若“Input Image Format”取值为YUV420 sp、YVU420 sp、YUV422 sp、YVU422 sp、BGR package，则“Model Image Format”取值为RGB、BGR。
  - 若“Input Image Format”取值为RGB 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。

  - 若步骤2中“Aipp Configuration File”选择的AIPP配置文件是读不出RGB数据的异常文件，则“Model Image Format”默认取值为GRAY。

- Crop：抠图开关开启后，参数解释以及抠图约束如下所示。
  - 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6-3“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6-3Input Nodes中shape取值的高和宽（模型文件input shape中的高和宽）。
- Mean：根据“Model Image Format”参数的取值，该参数的具体显示和值如下所示。
  - 若“Model Image Format”为RGB时，则该参数显示为Mean: [R][G][B]，每个通道的默认值为104、117、123，可根据需要手动修改默认值。
  - 若“Model Image Format”为BGR时，则该参数显示为Mean: [B][G][R]，每个通道的默认值为104、117、123，可根据需要手动修改默认值。
  - 若“Model Image Format”为GRAY时，则该参数显示的默认值为104，可根据需要手动修改默认值。
  - 若“Model Image Format”为RGBA时，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，可根据需要手动修改默认值。
  - 若“Model Image Format”为BGRA时，则该参数显示为Mean: [B][G][R][A]，每个通道的默认值为104、117、123、0，可根据需要手动修改默认值。
- Min：根据“Input Image Format”参数的取值，该参数的具体显示和值如下所示。
  - 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。

- 1/Variance：根据“Input Image Format”参数的取值，该参数的具体显示和值如下所示。
  - 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1/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1/Variance: [R][G][B][A]，每个通道的默认值都为1.0。
- Max Image Size (Byte)：根据图像输入格式的不同，该参数的取值也不同，该参数的具体取值如下所示。
  - 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。

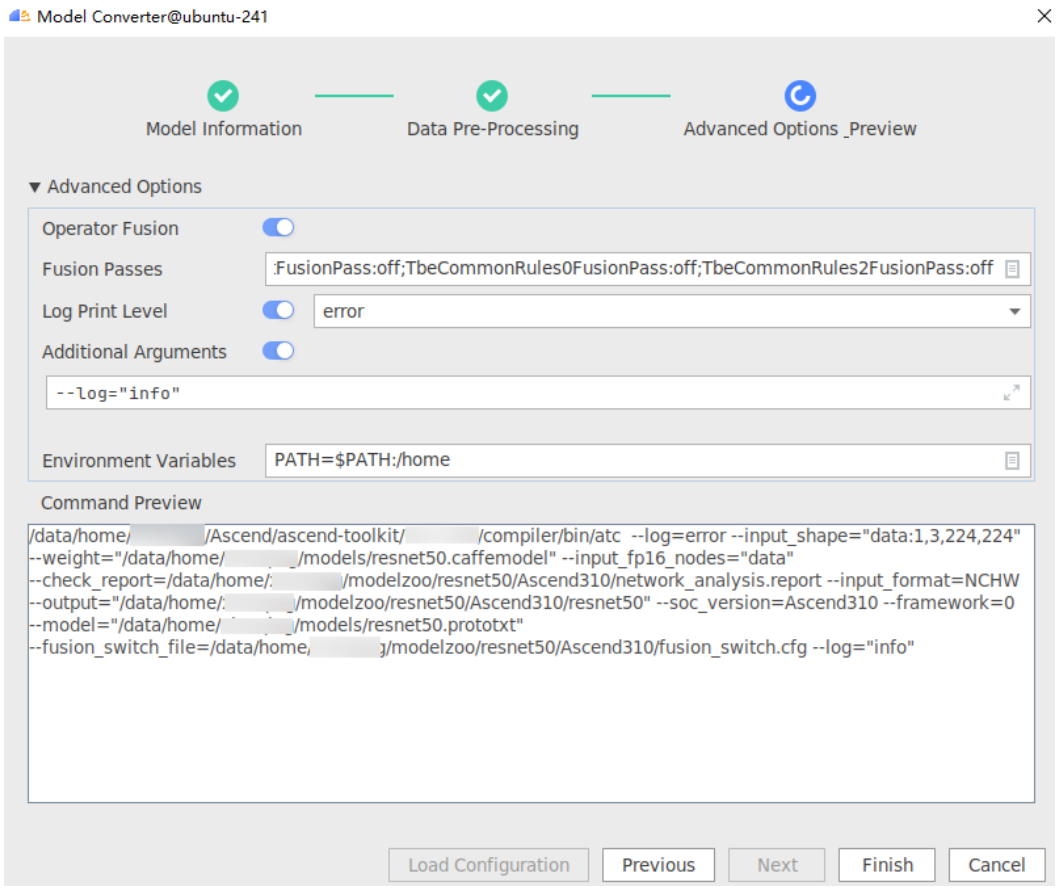
须知

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，界面参考如**图6-6**所示。

图 6-6 高级选项配置



参数说明如表6-4所示。

表 6-4 Advanced Options Preview 界面参数配置

参数	说明	备注
Operator Fusion	是否关闭融合功能。 • 打开表示关闭融合功能。打开该项将显示“Fusion Passes”。 • 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。 **说明** 关闭融合功能时，仅关闭配置文件中指定的融合规则，当前可以关闭的融合规则请参见《**图融合和UB融合规则参考**》，但是由于系统机制，其他融合规则无法关闭。

参数	说明	备注
Fusion Passes	需要关闭的融合规则。	打开“Operator Fusion”功能，会显示该项。默认关闭的融合规则以及如何添加需要关闭的融合规则详情请参见**•Fusion Passes：默认关闭的融合规则...**。
Log Print Level	日志级别配置开关，可以控制是否打印模型转换过程中对应级别的日志信息。 • 打开表示可以设置日志级别并打印对应级别的日志信息； • 关闭表示不打印日志信息。参数默认关闭。	日志级别： • error：输出error/event级别的运行信息。 • warning：输出warning/error/event级别的运行信息。 • info：输出info/warning/error/event级别的运行信息。 • debug：输出debug/info/warning/error/event级别的运行信息。
Additional Arguments	扩展转换参数。	• 模型转换界面不支持配置，但是ATC或者AOE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或者AOE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**》的“参数说明”章节或者《**AOE工具使用指南**》的“AOE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模型转换过程中日志的级别参数“Log Print Level”，对应ATC工具中的“--log”参数，则在“Additional Arguments”再次配置“--log”参数为其他数值，例如配置为“--log="info"”，则“Log Print Level”参数中指定的模式不生效。

参数	说明	备注
Environment Variables	环境变量。可选参数，请根据实际自行配置。	• 在文本框中添加环境变量。*环境变量_1=值1；环境变量_2=值2*，多个环境变量用英文分号隔开。 例如： TE_PARALLEL_COMPILER=8;REPEAT_TUNE=False 环境变量解释： – TE_PARALLEL_COMPILER：AOE场景下，算子编译所需环境变量。 – REPEAT_TUNE：AOE场景下，是否重新发起调优所需环境变量。 • 单击文本框后的 图标，在弹出的对话框中单击 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：*值1*。 环境变量的详细配置请参见《**AOE工具使用指南**》的“离线推理场景下调优（其他推理设备）> 配置环境变量”章节。
Command Preview	模型转换和调优使用的ATC或者AOE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或者AOE命令的结果预览，例如“Log Print Level”参数配置为“error”，则“Command Preview”区域展示的atc命令为“--log=error”。 • 如果“Additional Arguments”中再次配置界面已有的参数，例如配置为“--log="info"”，则“Command Preview”区域会追加相应的参数，模型转换和调优时，“Command Preview”区域后面的参数取值会覆盖前面已有的参数。

Advanced Options Preview界面参数详细说明：

- Fusion Passes：默认关闭的融合规则以及如何添加需要关闭的融合规则如下所示。
  - 默认关闭的融合规则：

    V100RequantFusionPass、V200RequantFusionPass、ConvConcatFusionPass、SplitConvConcatFusionPass、TbeEltwiseQuantFusionPass、TbeConvDequantVaddReluQuantFusionPass、TbeConvDequantVaddReluFusionPass、TbeConvDequantQuantFusionPass、TbeDepthwiseConvDequantFusionPass、TbeFullyconnectionElemwiseDequantFusionPass、TbeConv2DAddMulQuantPass、TbePool2dQuantFusionPass、TbeAippConvReluQuantFusion、TbeCommonRules0FusionPass、TbeCommonRules2FusionPass。
  - 添加需要关闭的融合规则：
    - 在文本框中输入需要关闭的融合规则。*融合规则名称1:off；融合规则名称2:off*，多个融合规则之间用英文分号隔开。
    - 单击文本框后的 图标，在弹出的对话框中单击 填写。
      - 在Name中输入融合规则名称：*融合规则名称1*。
      - 在Value中输入环境变量值：*off*。

      说明

      用户需要确保输入的融合规则的正确性。

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。

图 6-7 模型转换成功

步骤5 模型转换完毕，在服务器后台路径“$HOME/modelzoo/*resnet50*/$Soc_Version ”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和日志文件ModelConvert.txt。

- 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。
- 如果开启日志级别配置功能，当模型转换完毕后，无论转换成功或者失败，在MindStudio界面右下方都会弹出“ModelConvert Detail”弹窗，如下图所示。

图 6-8 转换成功弹窗

图 6-9 转换失败弹窗

- 单击“show model”可以在MindStudio界面打开转换成功后的模型可视化界面，模型可视化界面的操作方法请参见**可视化界面说明**。
- 单击“show log”可以在MindStudio界面打开日志文件ModelConvert.txt，显示的日志内容由“Log Print Level”参数选择的级别类型决定。

当Mode Type为ATC时，模型转换的日志文件（ModelConvert.txt）所在路径为：“$HOME/modelzoo/resnet50/$Soc_Version”。回显信息示例如下：

```
drwxr-x--- 2      4096 Mar 10 16:46 ./
drwx------ 3      4096 Mar 10 16:45 ../
-rw------- 1       127 Mar 10 15:55 fusion_switch.cfg        --融合开关配置文件
-rw------- 1       453 Mar 10 16:45 insert_op.cfg            --数据预处理配置文件
-rw-r----- 1       453 Mar 10 16:45 ModelConvert.txt         --日志文件
-rw------- 1      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  51581408 Mar 10 16:46 resnet50.om             --上板运行的模型文件
```

当Mode Type为AOE时，模型调优的日志文件（ModelConvert.txt）所在路径为：“$HOME/modelzoo/resnet50/aoe_type$Job Type”。回显信息示例如下：

```
dggphicprd32833:~/modelzoo/resnet50/aoe_type1$ ll
total 51416
drwxr-x--- 3      4096 Feb 24 16:16 ./
drwxr-x--- 4      4096 Feb 24 16:10 ../
-r-------- 1       475 Feb 24 16:16 aoe_result_sgat_20230224161101211040_pid57522.json      --调优结果文件（子图调优是sgat，算子调优是opat）
drwxr-x--- 3      4096 Feb 24 16:11 aoe_workspace/
-rw-r----- 1      6732 Feb 24 16:15 fusion_result.json
-rw------- 1       454 Feb 24 16:10 fusion_switch.cfg
-rw------- 1       577 Feb 24 16:10 insert_op.cfg
-rw-r----- 1    849212 Feb 24 16:16 ModelConvert.txt
-rw------- 1      3025 Feb 24 16:16 resnet50_config.json
-rw------- 1  51752967 Feb 24 16:15 resnet50.om
```

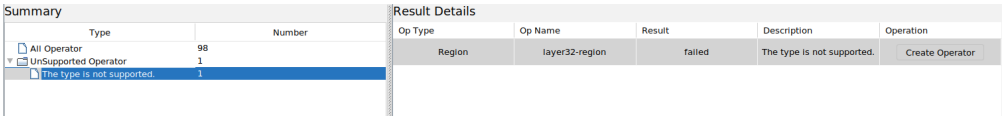
----结束

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6-10**所示整网支持度评估报告。

图 6-10 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。

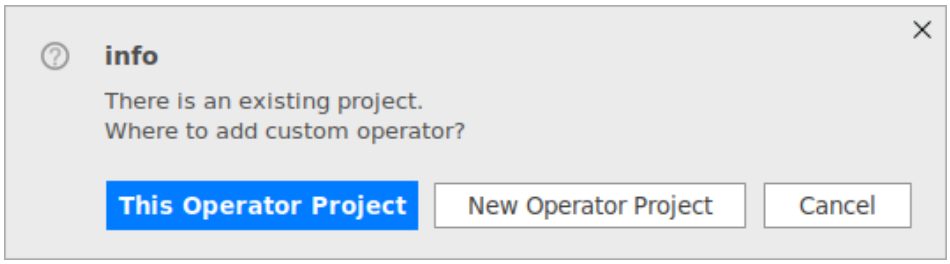
- **解决方法**

  a. 在图6-10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6-11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8.5 创建算子工程。

  图 6-11 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
├── build                          //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 算子开发。

  自定义算子开发完成后，重新进行模型转换。

6.5 模型可视化

对于原始模型文件（.pb、.onnx、.prototxt）或转换成功的.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。下文中以.om模型文件为例展示模型可视化功能。

前提条件

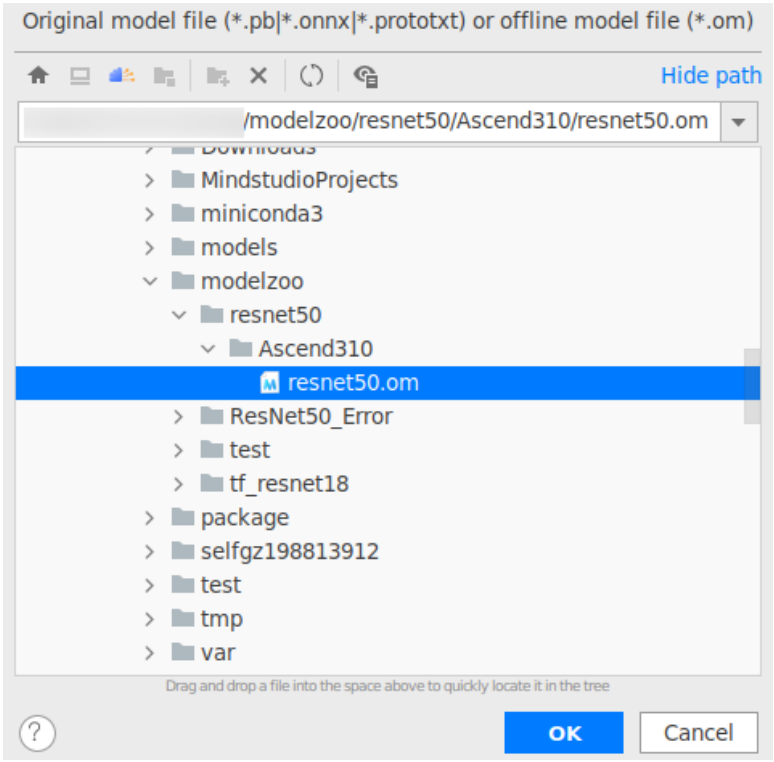
如果Linux环境为aarch64架构，且系统中的glibc版本小于2.29，则无法使用MindStudio的模型可视化功能，处理方法请参考**14.3.25 glibc版本过低导致无法使用模型可视化功能**。

操作步骤

步骤1 依次单击菜单栏“Ascend > Model Visualizer”，或在工具栏选择，在弹出窗口中选择要可视化的模型，例如此处的resnet50模型文件。

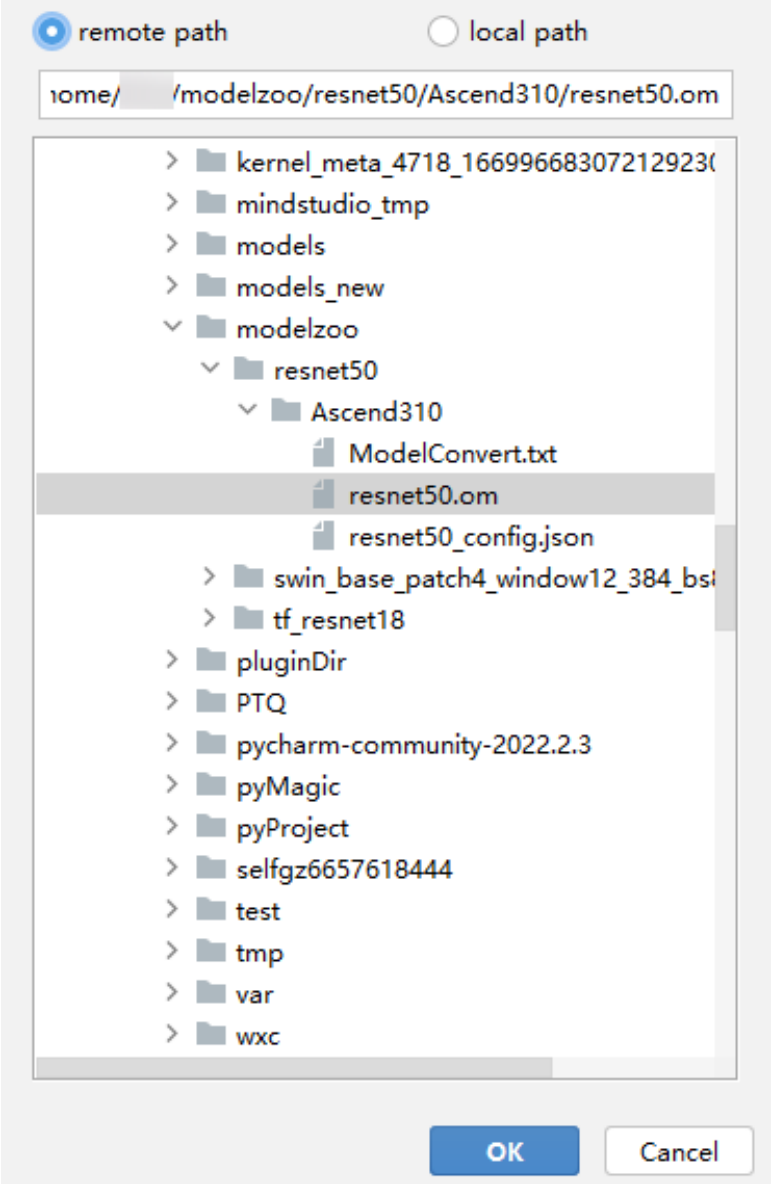
- 在Linux环境上使用模型可视化功能时，如**图6-12**所示。

图 6-12 选择要打开的模型（Linux）



- 在Windows环境上使用模型可视化功能时，如**图6-13**所示。

图 6-13 选择要打开的模型（Windows）

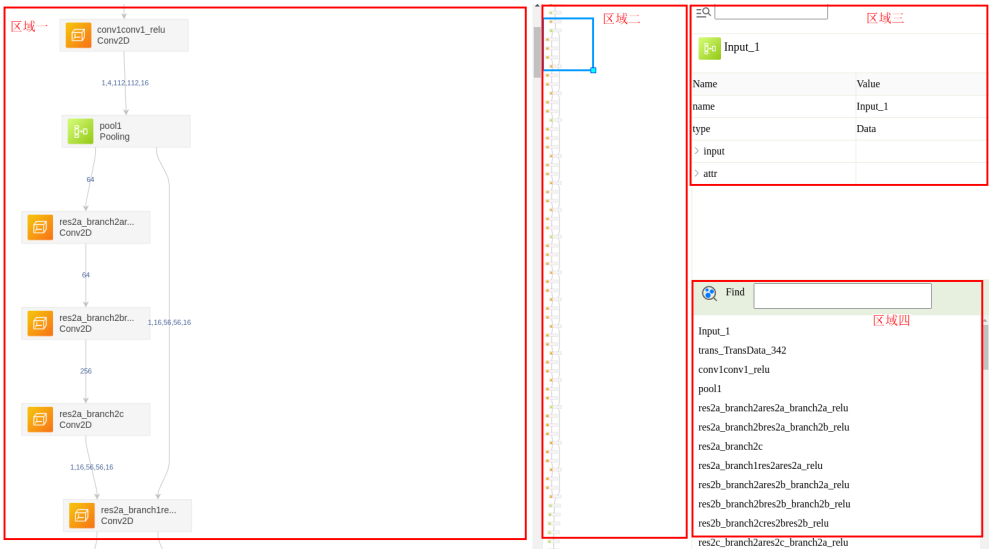


说明

查看.prototxt模型文件时，需要保证.prototxt模型文件与其相关的.caffemodel权重文件在同一目录下。

步骤2 鼠标左键双击resnet50.om模型文件或者单击“OK”，打开模型可视化界面如图6-14所示。

图 6-14 模型可视化界面



----结束

可视化界面说明

- **查看算子信息**

  图6-14区域一中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，**区域三**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。

  区域2会展示该网络模型的整体结构，包括**区域三**中展示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。

  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。

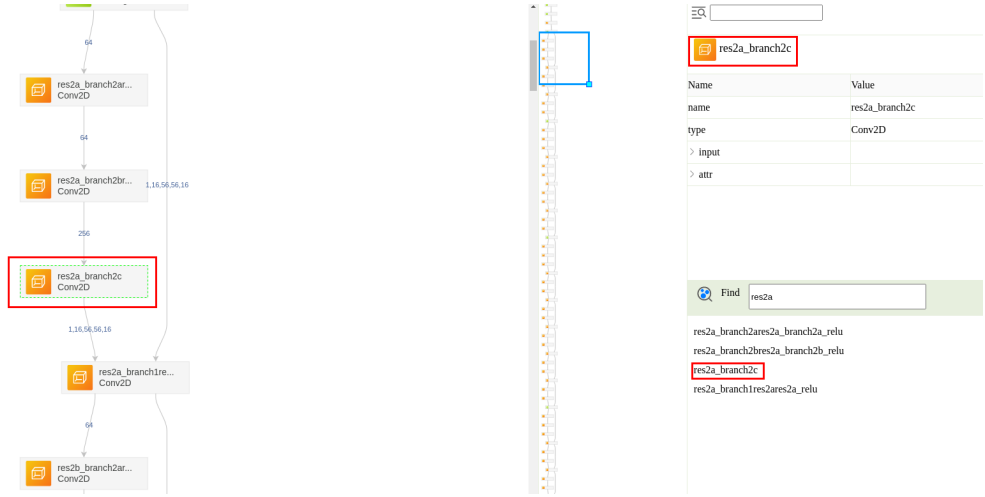
- **查看算子输出维度和shape信息**

  图6-14区域一中还展示了每一层算子输出的shape信息，如每一层算子连接线中间的1,224,224,4等shape信息。

- **搜索算子**

  图6-14区域四中的“Find”输入框中输入算子名称，下方搜索区域会列出相关的算子。选择其中一个算子，**区域一**中的网络拓扑结构中相应算子会显示绿色选中框，**区域三**会展示该算子的详细信息。如**图6-15**所示。

图 6-15 模型可视化界面的搜索功能



- **搜索算子内部信息**

  选**区域一**中某层算子，在**区域三**图标后的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，然后单击图标，如果匹配到相关信息，则**区域三**中相关信息会高亮显示，否则会在“Output”窗口提示“Value not found”。

- **查看模型输出节点信息**

  当模型转换时通过Select节点设置了输出算子，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，可以单击相应的算子，查看其输入信息。

- **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**6.4 执行转换**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，用户可以单击对应的节点，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，单击对应的节点，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。

其他操作

Linux环境aarch64架构

UbuntuOS：https://mindstudio-sample.obs.cn-north-4.myhuaweicloud.com/glibc/euler-os/glibc-2.29.tar.gz

EulerOS：https://mindstudio-sample.obs.cn-north-4.myhuaweicloud.com/glibc/euler-os/glibc-2.29.tar.gz

7 应用开发

7.1 功能简介

MindStudio支持基于AscendCL（Ascend Computing Language）或MindSDK开发应用。

MindSDK致力于简化昇腾芯片推理业务开发过程，降低使用昇腾芯片开发的门槛。

使用AscendCL提供的C语言API库开发深度神经网络应用，在昇腾CANN平台上开发应用，实现深度学习推理计算、图形图像预处理、单算子加速计算等。

7.2 基于 MindSDK 开发应用

7.2.1 开发前须知

- 目前MindStudio仅支持MindSDK中的Vision SDK，暂不支持Index SDK。
- Vision SDK当前适用于UbuntuOS、CentOS、EulerOS、LinxOS、KylinOS等系统，软件包运行版本兼容情况请参见《**Vision SDK 用户指南**》。
- **对于运行环境在远端的Python版本应用工程**，在应用开发开始前需要对运行环境进行配置。

  ```
  LD_LIBRARY_PATH=$HOME/Ascend/ascend-toolkit/{version}/runtime/lib64:$LD_LIBRARY_PATH;PYTHONPATH=$HOME/Ascend/ascend-toolkit/{version}/python/site-packages/acl:$PYTHONPATH
  ```

  其中*{version}*内容请根据实际情况填写，可选择通过以下任意一种方式处理。
  - 修改运行环境中的“.bashrc”文件，加入“PYTHONPATH”与Runtime路径。
  - MindStudio编译/运行配置时，在“Run/Debug Configuration”功能中添加“Environment Variables”，补充环境变量，详细操作请参见**7.2.7 编译与运行应用工程**。

- Vision SDK运行依赖Python 3.9，请在安装CANN前确保Python 3.9已安装，并完成Python 3.9的环境变量配置。
- 请确保运行环境已安装driver包，在启动MindStudio之前，需在运行环境上手动执行Ascend-cann-toolkit与MindSDK安装目录中的“set_env.sh”设置环境变量，可参照以下方式执行，软件安装路径请以实际为准。（其中*{version}*为SDK软件包版本，*{arch}*为系统架构。）

```
source $HOME/Ascend/mindx_sdk/mxVision_{version}/linux-{arch}/mxVision/set_env.sh
source $HOME/Ascend/ascend-toolkit/set_env.sh
```

7.2.2 安装 MindSDK 软件包

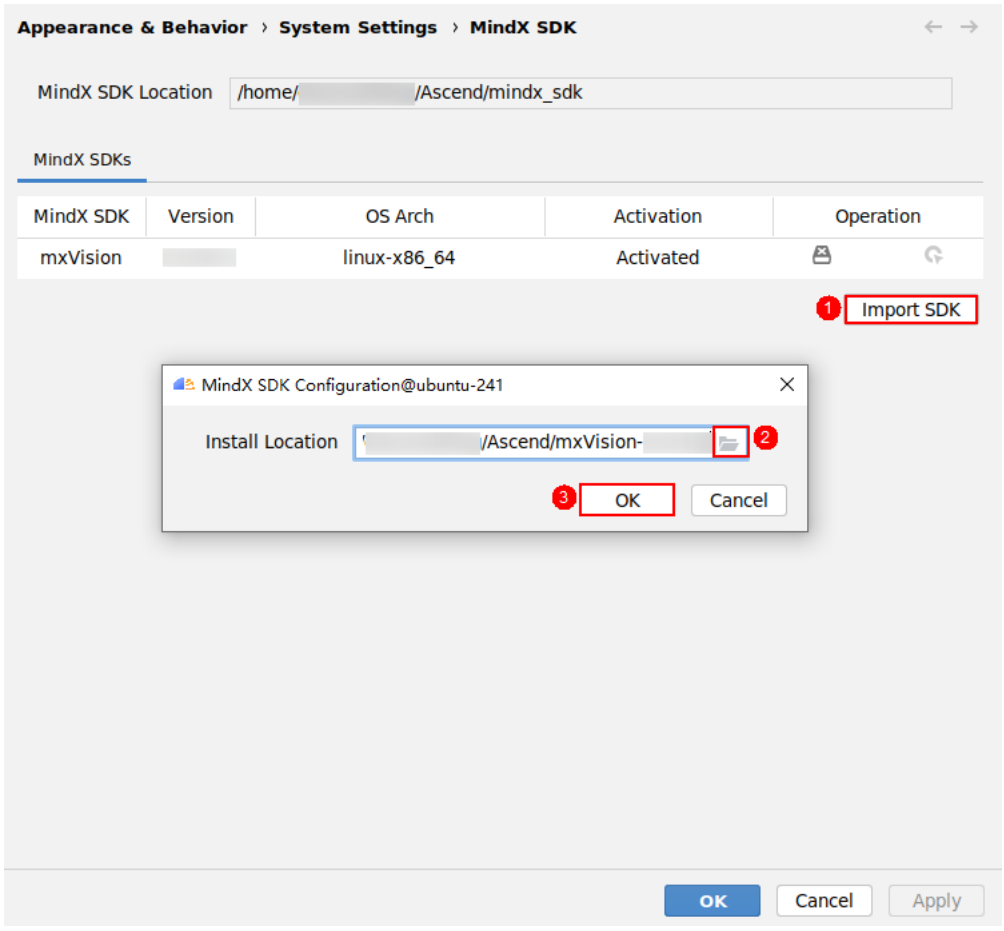
7.2.2.1 Linux 场景安装

步骤1 参考《Vision SDK 用户指南》“安装部署”章节，安装Vision SDK。

步骤2 工程创建成功或打开已有的工程后，在顶部菜单栏中单击“File > Settings”并在弹出的窗口左侧导航栏中找到“Appearance & Behavior > System Settings > MindX SDK”，进入MindSDK管理界面。

步骤3 单击界面中的“Import SDK”并在弹出的界面中单击右侧，选择MindSDK软件包安装目录，如图7-1（请选至版本号一级），单击“OK”，完成导入。

图 7-1 导入已安装的 MindSDK



----结束

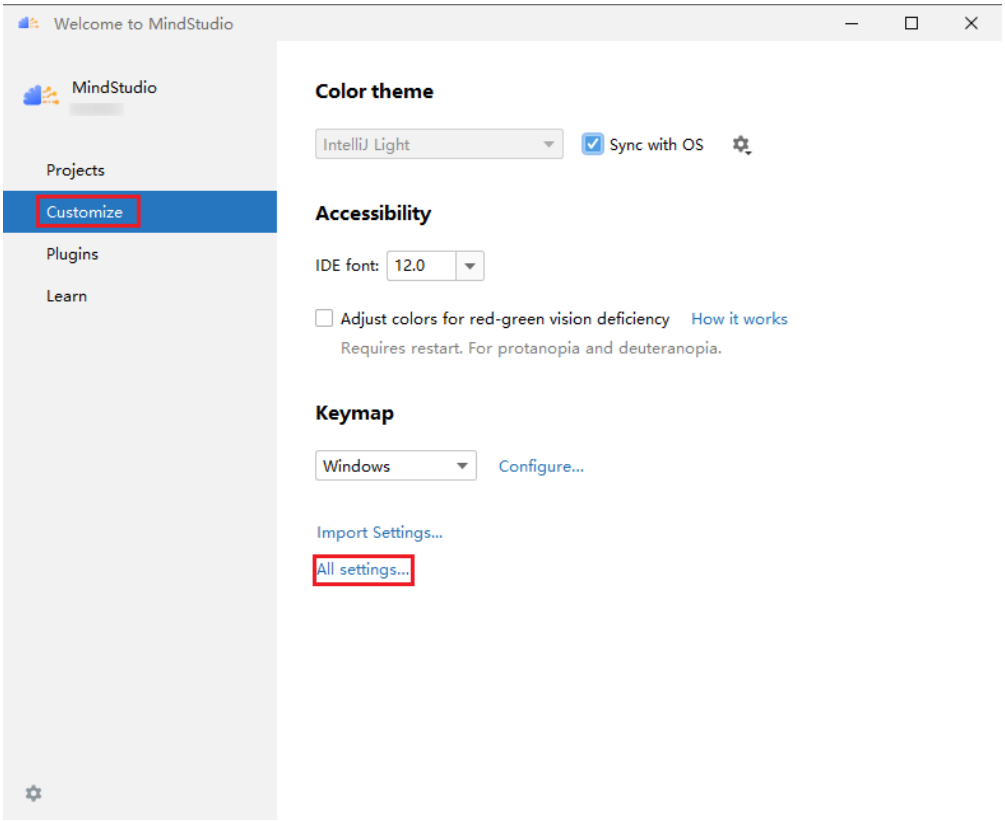
7.2.2.2 Windows 场景安装

步骤1 请参见《Vision SDK 用户指南》的“安装部署”章节，安装MindSDK软件包。

步骤2 可通过以下方式进入MindSDK管理界面。

- 如当前未打开任何工程，在欢迎界面左侧导航栏，如图7-2，选择“Customize”一栏，并在右侧功能界面中找到并单击“All settings”。在打开的“Settings”界面左侧导航菜单找到“Appearance & Behavior > System Settings > MindX SDK”，并在右侧的功能界面单击“Install SDK”进入MindSDK管理界面。

  图 7-2 通过欢迎界面打开 Settings



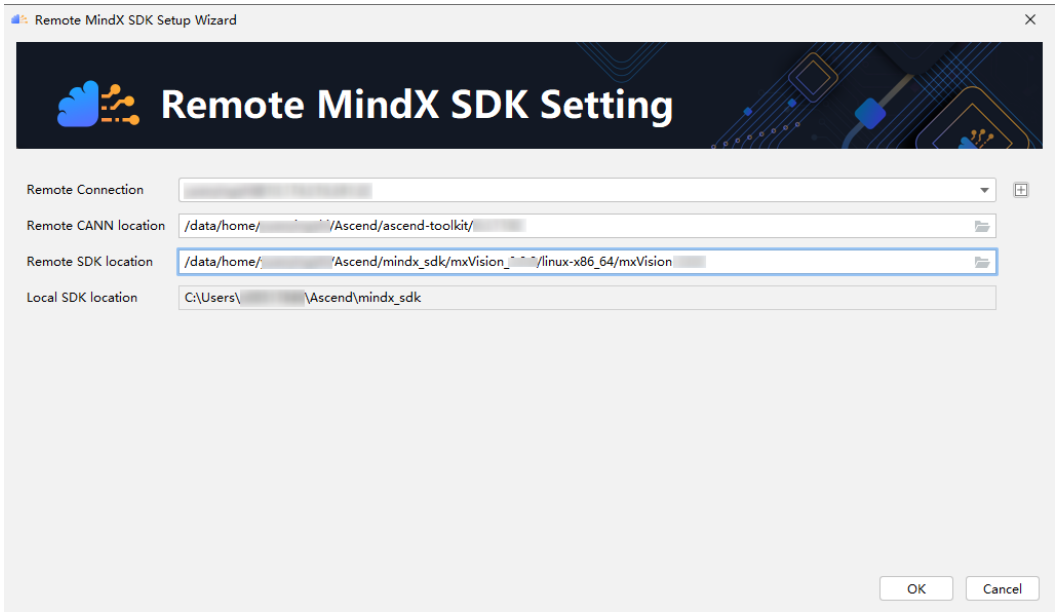
- 在Windows本地进入已完成创建的工程页面，在顶部菜单栏中单击“File > Settings”，在打开的“Settings”界面左侧导航菜单找到“Appearance & Behavior > System Settings > MindX SDK”进入MindSDK管理界面。界面中“MindX SDK Location”为软件包的默认安装路径，默认安装路径为“C:\Users*用户名*\Ascend\mindx_sdk”。单击“Install SDK”进入MindSDK管理界面。

步骤3 在MindSDK管理界面，如图7-3，完成远端CANN开发套件包、MindSDK软件包安装目录配置，单击“OK”结束，返回SDK管理界面。

- “Remote Connection”：远程连接的用户及IP，可通过右侧  进行添加，具体配置添加操作请参见13.2 SSH连接管理。
- “Remote CANN Location”：远端环境上CANN开发套件包的路径，请配置到版本号一级。
- “Remote SDK Location”：远端环境上SDK的路径，请配置到版本号一级。IDE将同步该层级下的“include”、“opensource”、“python”、“samples”文件夹到本地Windows环境，层级选择错误将导致安装失败。

- “Local SDK Location”：同步远端环境上SDK文件夹到本地的路径。默认安装路径为“C:\Users*用户名*\Ascend\mindx_sdk”。

图 7-3 MindSDK 管理界面



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

界面中两个按钮介绍如下：

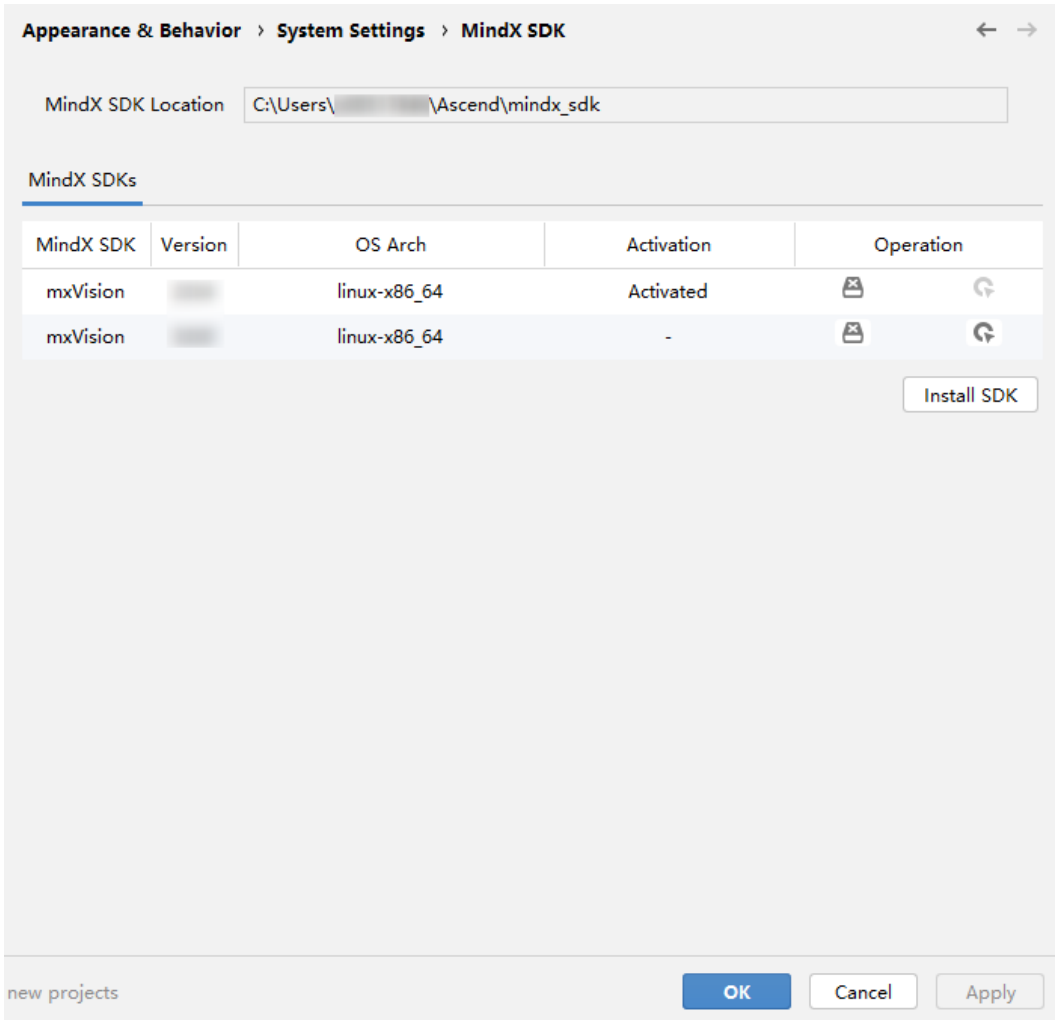
-  ：卸载按钮。
-  ：激活按钮。

说明

对激活按钮说明如下：

- 所有已安装版本的软件中，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 MindSDK 管理界面



----结束

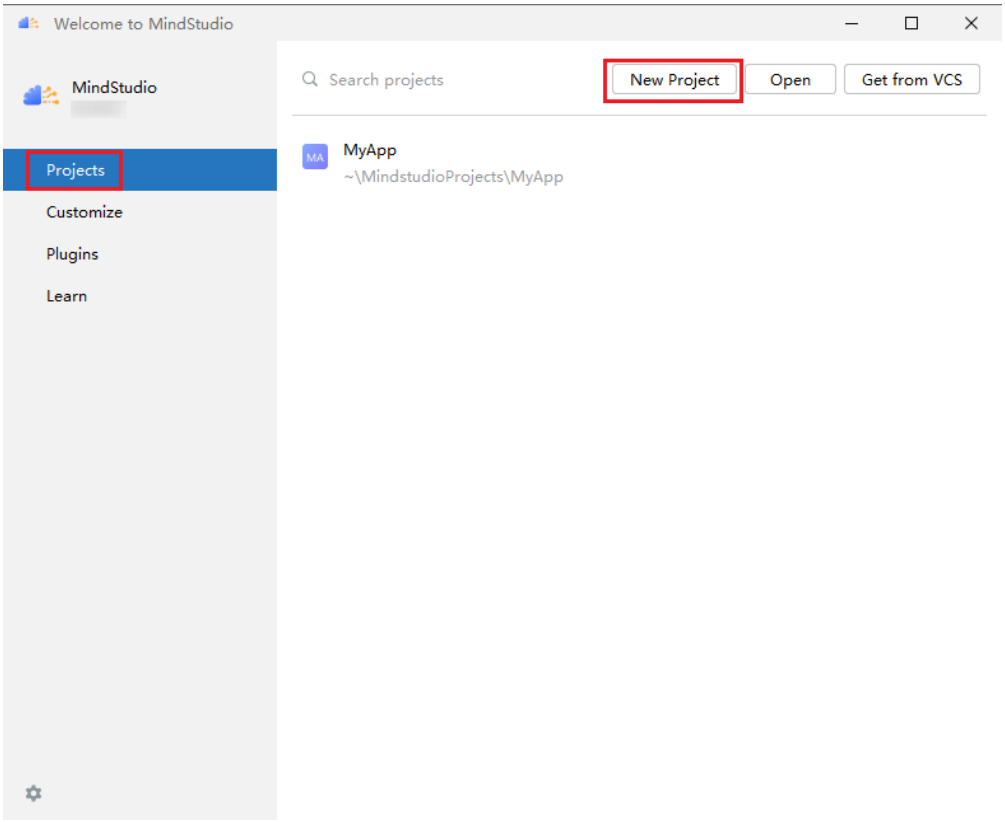
7.2.3 创建应用工程

新建模板工程

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

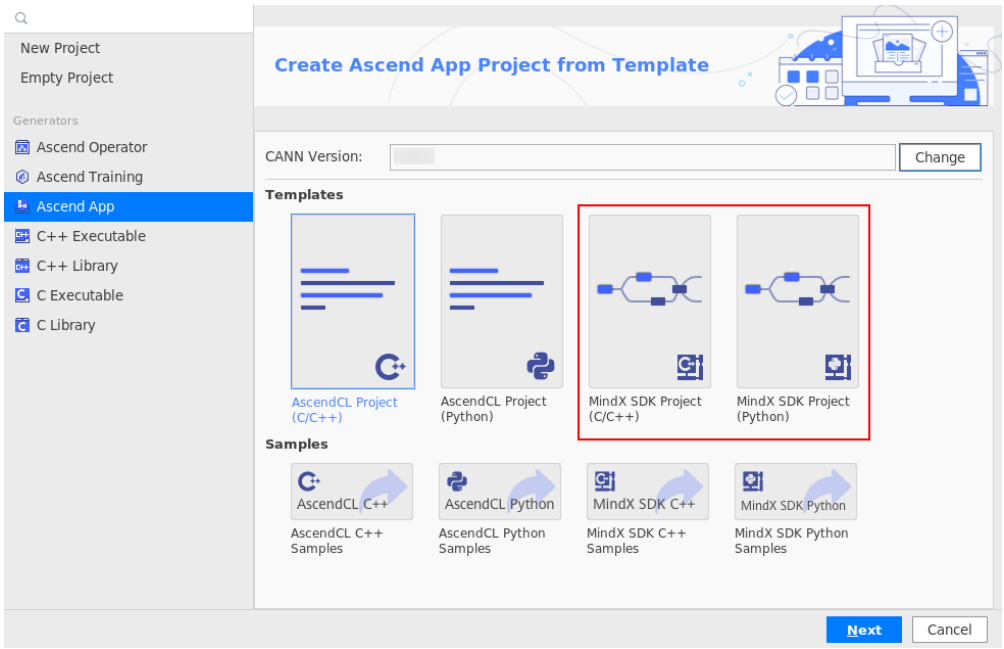
图 7-5 工程创建界面



- MindStudio工程界面：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”，选择工程类型，如图7-6。

图 7-6 MindSDK Project 模板工程



CANN Version：当前激活的CANN版本，可通过单击右侧“Change”进行变更，具体功能使用请参见13.6.2 切换/激活CANN包。

MindX SDK Project(C/C++)和MindX SDK Project(Python)为MindSDK空白工程，仅包括开发框架，不含具体的代码逻辑。

说明

如未在MindStudio中配置CANN，部分应用工程可能无法正常显示。

步骤3 点击“Next”进入下一步，配置工程相关参数如表7-1所示。

表 7-1 工程参数说明

参数	说明
Project name	工程名称，自行配置。 名称开头和结尾必须是数字或字母。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。
Project location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）
More Settings	“Module name”：模块名，默认与“Project name”一致。
	“Content root”：根目录下路径。
	“Module file location”：模块文件路径。
	单击“Project format”右侧选框，出现下拉菜单。 • .idea（directory-based）：创建项目时生成一个.idea项来保存项目的信息，默认选项。 • .ipr（file-based）：项目配置文件来保存项目的配置信息。

步骤4 单击“Create”，完成工程创建。成功创建MindSDK Project模板工程后，工程目录的主要结构如下，请以实际创建结果为准。

- MindX SDK Project(C++)：

```
├── 工程名
│   ├── build                  //存放cmake依赖文件。
│   │   └── cmake
│   ├── CMakeLists.txt         //编译脚本，调用src目录下的CMakeLists文件。
│   ├── config
│   │   ├── logging.conf       //日志配置文件。
│   │   └── sdk.conf
│   ├── out                    //存放编译出的可执行文件。
│   │   └── config
│   └── 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。
```

- MindX SDK Project(Python)：

```
├── 工程名
│   ├── config                 //存放配置文件。
│   │   ├── logging.conf
│   │   └── sdk.conf
│   ├── .idea
│   │   ├── misc.xml
│   │   ├── modules.xml
│   │   └── workspace.xml
```



```
├── main.py              //运行入口。
├── 工程名.iml
└── .project
```

----结束

导入应用工程

步骤1 导入工程文件，可通过选择以下任意一种方式完成。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的  ，选择现有工程打开。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in <*工作区目录*>”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤3 成功导入工程后，工程目录以树状呈现，请以实际创建结果为准。

----结束

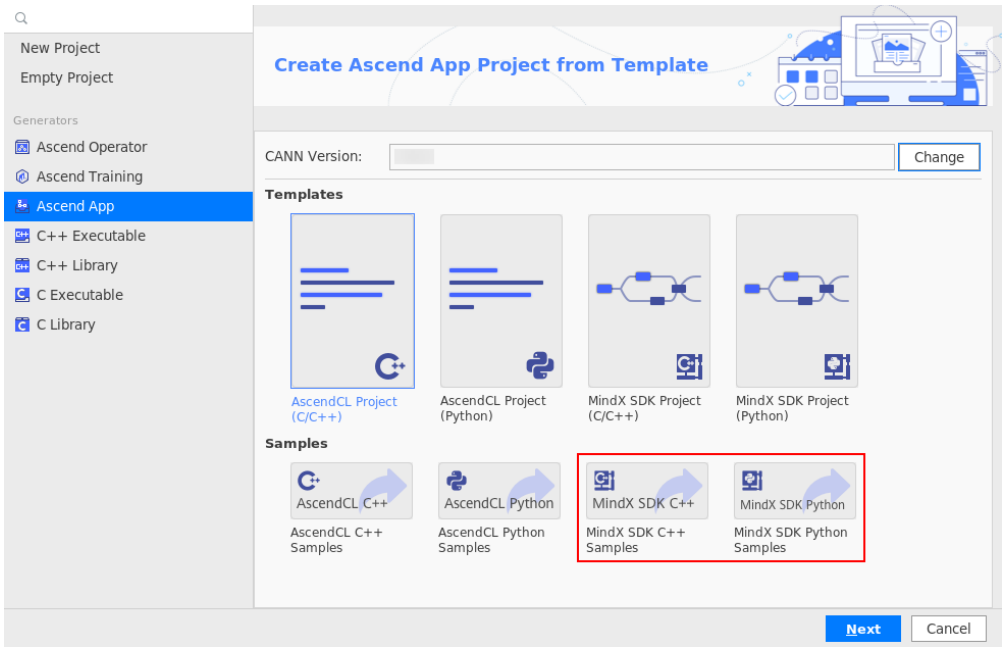
新建样例工程

步骤1 进入工程创建页面。

- MindStudio欢迎界面：单击“New Project”。
- MindStudio工程界面：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”，选择工程类型，如图7-7。

图 7-7 MindSDK Samples 样例工程



CANN Version：当前激活的CANN版本，可通过单击右侧“Change”进行变更，具体功能使用请参见**13.6.2 切换/激活CANN包**。

MindX SDK C++ Samples和MindX SDK Python Samples为基于MindSDK开发的样例工程。

说明

如未在MindStudio中配置CANN，部分应用工程可能无法正常显示。

步骤3 单击“Next”，浏览器会跳转至对应的Gitee代码仓界面。

步骤4 在Gitee代码仓页面下单击“克隆/下载 > 复制”，复制代码包下载链接。

步骤5 在开发环境执行命令：git clone URL（其中URL为复制的代码包下载链接），直接将代码包克隆到开发环境。

```
git clone https://gitee.com/ascend/mindxsdk-referenceapps.git
```

步骤6 在下载的文件夹中选择需要的样例，然后直接通过MindStudio导入，详情请参见**导入应用工程**。

----结束

7.2.4 开发应用

7.2.4.1 典型业务流程

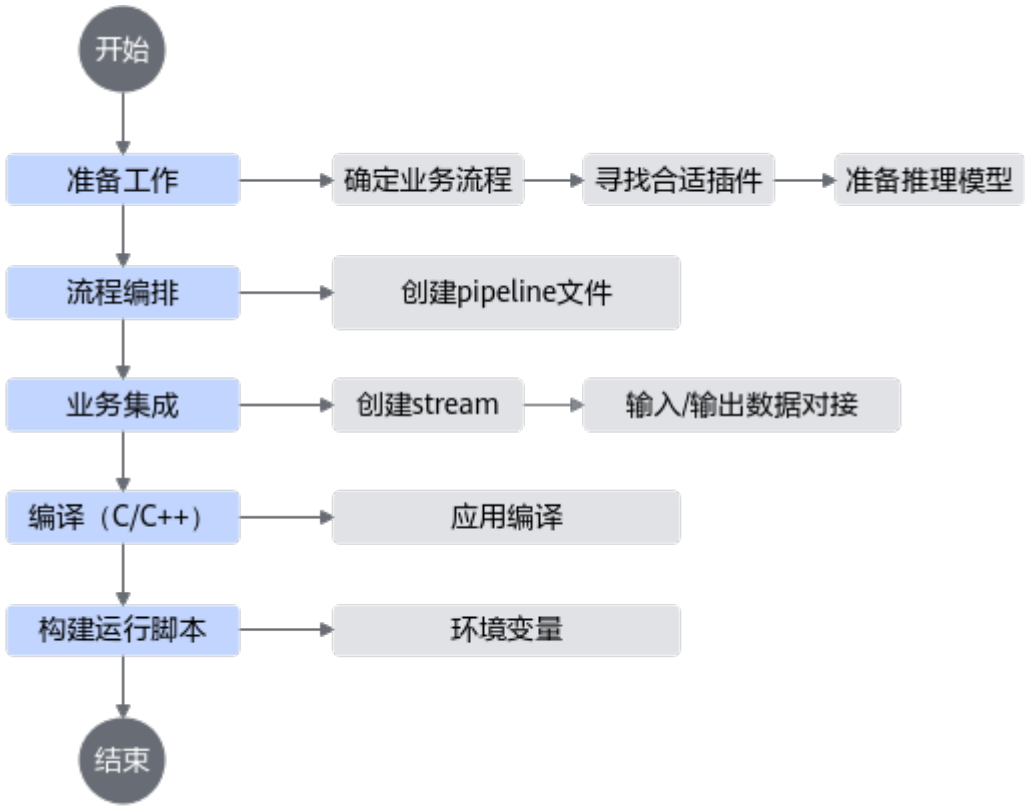
典型推理业务流程如**图7-8**所示，先通过目标检测获取目标坐标，然后通过图像分类识别目标属性。流程中有两次图像缩放的操作，两者主要区别在于设置的缩放宽高参数不同。

图 7-8 典型推理业务流程

7.2.4.2 开发流程

使用MindStudio开发和运行推理业务步骤如图7-9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 7-9 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，请参见图7-8。

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。请参见7.2.6 插件开发、《Vision SDK用户指南》的“插件参考”章节。

步骤3 准备推理模型文件与数据集。

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。（加载的模型路径中不能有空格。）

1. 转换模型。

   在添加模型文件前，请参见**6 模型转换和调优**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）。模型转换具体指导与相关参数设置请参见《**ATC工具使用指南**》。

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**7.2.5 可视化流程编排**、《**Vision SDK 用户指南**》的“使用流程编排方式开发”章节。

步骤5 业务集成。

编写C++程序或Python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《**Vision SDK 用户指南**》的“API参考（C++）”章节与“API参考（Python）”章节。

步骤6 编译与运行应用。

请参见**7.2.7 编译与运行应用工程**。

说明

MindSDK昇腾应用工程支持Profiling功能，具体可参见**10 性能分析**。

----结束

7.2.5 可视化流程编排

MindSDK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，实现相应的功能。可视化流程编排通过可视化的方式，开发数据流图，生成pipeline文件供应用框架使用。

步骤1 进入工程创建页面，用户可通过以下方式开始流程编排。

- 在顶部菜单栏中选择“Ascend > MindX SDK Pipeline”，打开空白的pipeline绘制界面，如**图7-10**。
- 在顶部菜单栏中选择“File > Open...”，打开用户自行绘制的pipeline文件，如**图7-11**（若自行绘制pipeline文件已放入工程目录中，也可在工程目录栏直接双击该文件打开）。

绘制界面分为左侧插件库、中间编辑区、右侧插件属性展示区，左侧插件库与中间编辑区请见**图7-11**，请参见**表7-2**。

绘制界面右下方按键请参见**表7-3**。

图 7-10 pipeline 绘制界面

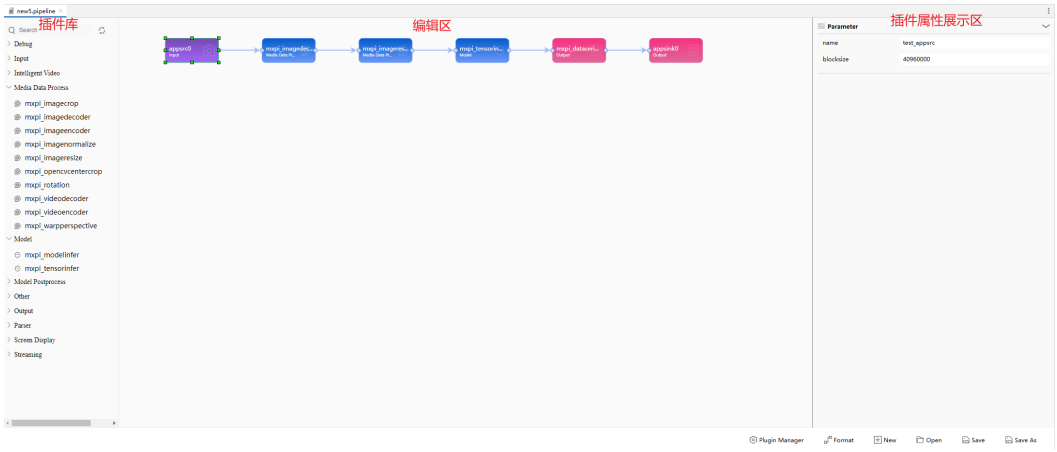


图 7-11 自行绘制 pipeline 文件

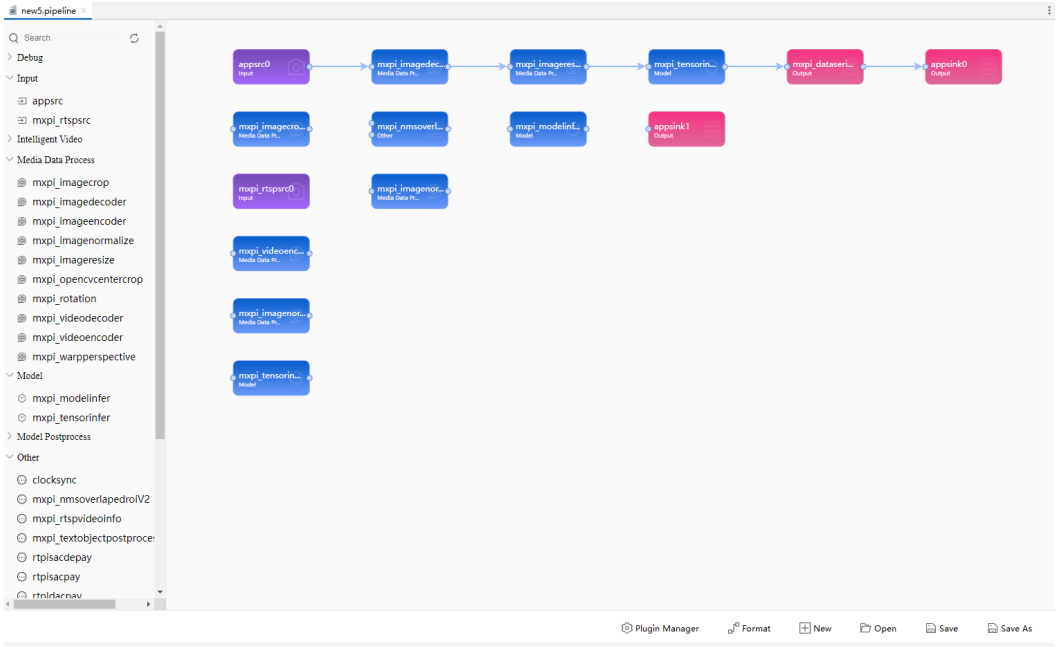


表 7-2 绘制界面介绍

区域	介绍
插件库	• 根据已安装的MindSDK扫描获取，在工程目录下生成“AllPluginsInfo.json”文件，MindStudio将会通过该文件识别插件以保证生成的pipeline可以找到对应的so。用户可将插件库目录展开，直接将所需插件拖动到编辑区内，或在搜索框中以关键字查找到插件，然后单击拖动到编辑区内。 • SDK开发套件有自带插件库，安装完成SDK开发套件后打开绘制界面，左侧插件库会自动加载，自带插件库可查询《**Vision SDK 用户指南**》的“插件参考”章节。 • 若自带插件库无法满足开发需求，用户可自行开发插件并通过**Plugin Manager**功能导入插件库，若目录不存在需要自行创建。开发插件的指导可参见**7.2.6 插件开发**、《**Vision SDK 用户指南**》的“使用流程编排方式开发”章节。 • 在工程开发过程中，如有插件变更，用户可通过插件库中的  按钮重新扫描刷新。
编辑区	• 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的连接关系。 • 每个插件的可配置的属性信息可被扫描，单击选中编辑区内的插件，用户可以在插件属性展示区自定义配置（如插件属性中的路径参数）。 **说明** – Remote场景mxpi_modelinfer插件中，选择完模型、标签和配置文件的输入路径后，请参照Linux系统手动同步改为Remote端的绝对路径，如模型文件路径修改为：“/home/workplace/*.om”，且保证所选路径有对应文件才能运行成功。 – “Name”字段为元件自定义名称，若用户修改后会以该字段内容作为元件名称进行使用，具体请查看《**Vision SDK 用户指南**》。 • 以**图7-11**为例，紫色为输入插件，红色为输出插件，其余插件为蓝色，灰色为当前插件库中未定义的插件。 • 鼠标悬停在插件上时，会显示该插件的功能描述信息。 • 单击键盘“Del”键可删除选中的插件或连线，在选中状态下，可对插件块进行大小拖放操作，最大尺寸可支持2倍宽度和2倍高度。 • 打开已有内容的pipeline文件，编辑区的左下方会有“Pipeline Stream Editor”与“Text”选项。支持图形化显示（默认）与代码显示两种方式，建议用户以图形化显示方式使用流程编排可视化功能。
插件属性展示区	• 单击编辑区中的插件时，会展示插件的属性。 • 插件属性以**key-value**的一对一简单形式或**subkey-subvalue**的复杂形式展现。 • **subkey-subvalue**形式下，单击  删除键值对，单击  新增键值对。

表 7-3 界面功能介绍

功能	作用
Plugin Manager	自定义插件管理，可通过该功能添加自定义插件的目录，导入用户的自定义插件库。（插件所在目录权限需要“440”权限，请在导入前进行确认。） 1. 单击“+ Add Plugin”按钮，在弹出的窗口中选中自定义插件so文件所在目录，单击“OK”。 2. 完成所有插件目录配置后单击“Save”，保存当前的自定义插件配置。 3. 打开pipeline，点击插件库中的 并根据提示重新打开pipeline，可通过插件库中查看到自定义插件已完成加载。 在使用Plugin Manager导入自定义插件时，请注意插件所需的依赖，如果插件存在未满足的依赖，该插件可能无法在插件库中显示。
Format	格式化按键，可一键解决插件的重叠或一键格式化整理用户绘制完成的插件流程。
New	新建画板。
Open	打开pipeline文件。
Save	以当前pipeline文件名称保存。
Save As	自定义保存绘制完成的pipeline插件流程图文件。单击弹出保存窗口。 参数介绍如下，用户可自行定义： • “Pipeline Filename”：pipeline文件的名称。 • “Output Path”：保存的路径。

步骤2 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，选中错误插件或者错误连线单击键盘“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

默认“Stream name”为“stream0”，“Device Id”为“0”。请根据实际需求进行修改，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。

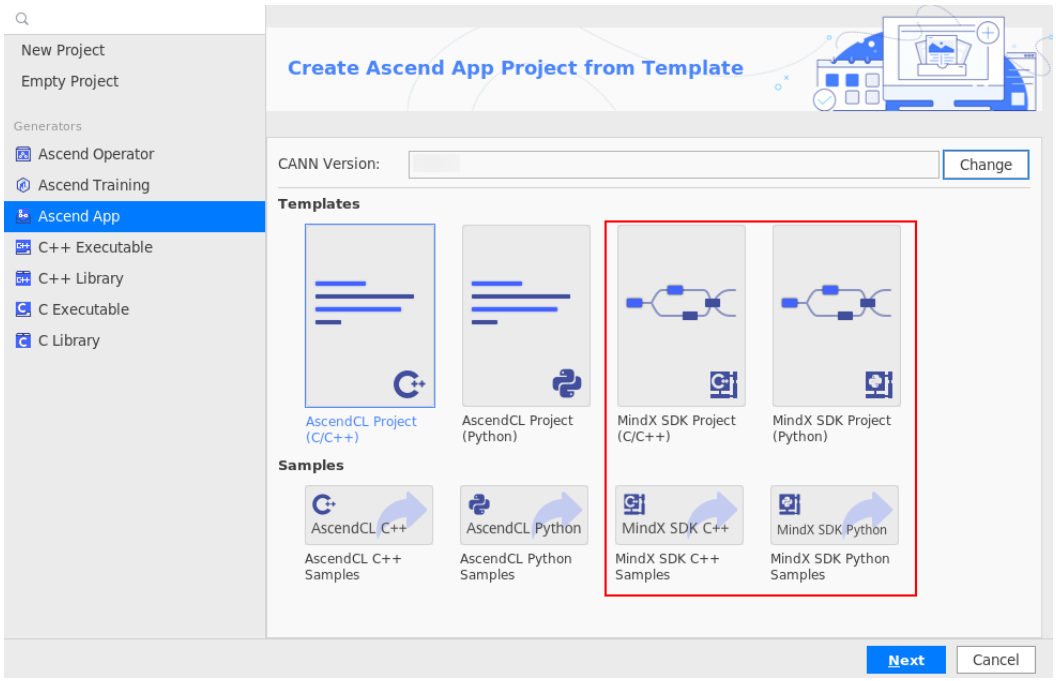
步骤3 绘制完成后，单击“Save”按钮保存。

----结束

7.2.6 插件开发

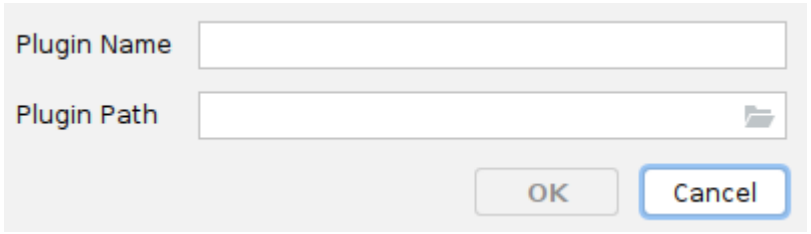
步骤1 如图7-12所示，点击新建空白工程MindSDK Project或从Gitee下载导入样例工程MindSDK Samples。

图 7-12 基于 MindSDK 开发的应用工程



步骤2 在IDE页面内的工程目录处右键工程名，选择“New MindX SDK Plugin”会出现如图7-13弹窗。

图 7-13 插件创建弹窗



- “Plugin Name”：插件名称，用户自行定义。
- “Plugin Path”：插件创建目录，建议放在工程下的“plugin”目录，若目录不存在请自行创建。

步骤3 创建完成后会在工程目录中生成如下目录层级，*Plugin Name*为图7-13中用户设定的插件名称。

```
└── plugin
    └── Plugin Name        // 插件模板。
        ├── Plugin Name.cpp
        ├── Plugin Name.h
        └── CMakeLists.txt
```

步骤4 在工程根目录下的“CMakeLists.txt”（图中标号“1”所示文件）或在“C++”目录中的“CMakeLists.txt”（图中标号“2”所示文件）新增插件的“CMakeLists.txt”（图中标号“3”所示文件）所在的位置，将新增插件加入编译过程。

以新增插件mxpi_sample为填写示例。

- 在“*工程文件根目录*”下的“CMakeLists.txt”中加入：
  ```
  add_subdirectory("./plugin/mxpi_sample")
  ```

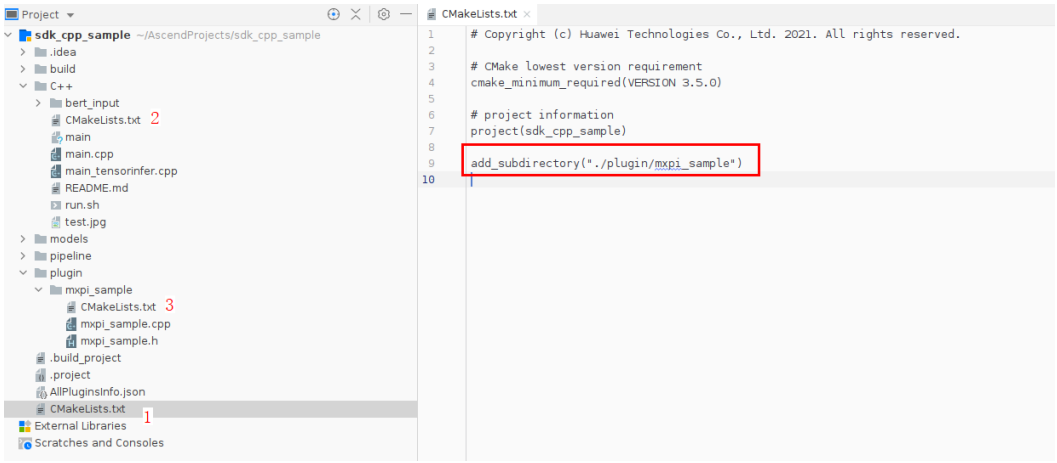
“./plugin/mxpi_sample”为新增插件的“CMakeLists.txt”相对于“*工程文件根目录*”下的“CMakeLists.txt”所在的路径。

- 在工程文件下C++目录下的“CMakeLists.txt”中加入：

```
add_subdirectory("../plugin/mxpi_sample")
```

“../plugin/mxpi_sample”为新增插件的“CMakeLists.txt”相对于在工程文件下C++目录下的“CMakeLists.txt”所在的路径。

图 7-14 add_subdirectory



步骤5 编译完成后的.so文件在“*工程文件根目录*/lib/plugins”生成，并将文件权限设置为“640”。

步骤6 请用户参照《**Vision SDK 用户指南**》的“使用流程编排方式开发”章节，完成插件开发。

----结束

7.2.7 编译与运行应用工程

7.2.7.1 Linux 场景编译运行

编译应用工程

说明

- 若新建的工程为Python版本的应用工程，由于不需要执行“编译应用工程”，在MindStudio工程界面中“Build”功能不可用。
- 远端编译时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“model”、“profiling”目录不会被拷贝。
- 编译前，用户需确认工程目录下“set_env.cmake”文件的环境变量，替换其中的CANN软件包安装路径和MindSDK软件包安装路径。

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})
  ```

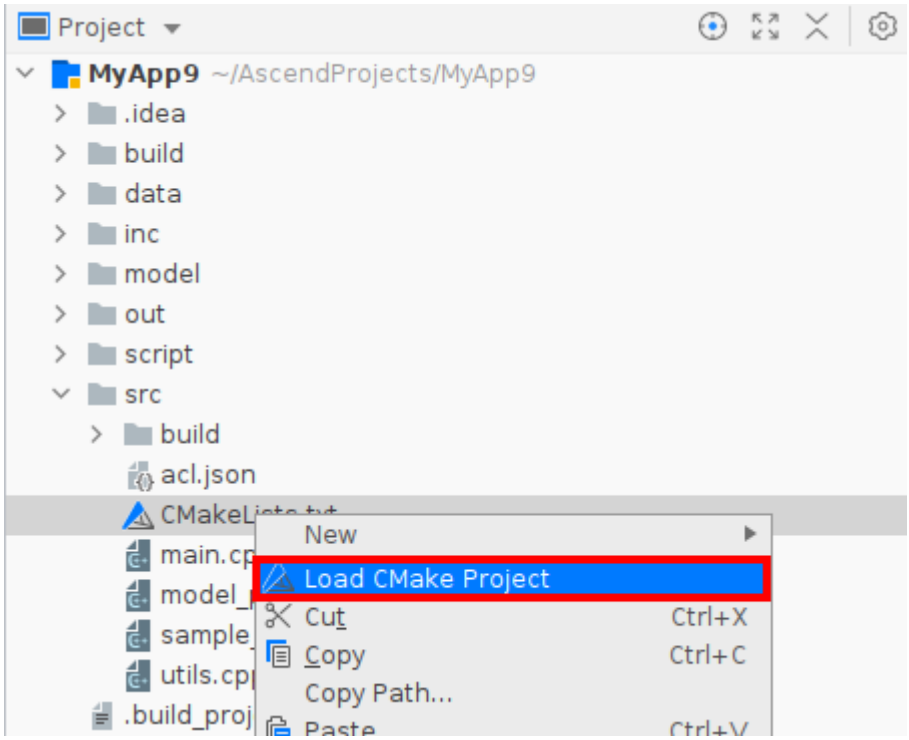
说明

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 （可选）指定“CMakeLists.txt”编译配置文件。

在工程界面左侧目录找到“CMakeLists.txt”文件，右键弹出并单击如**图7-15**所示“Load CMake Project”，即可指定此配置文件进行工程编译。

图 7-15 右键指定 CMakeLists.txt 编译配置文件



步骤3 编译配置。

在MindStudio工程界面，顶部菜单栏中单击“Build > Edit Configuration...”，进入编译配置页面。

1. 配置编译参数，单击 + 添加新增配置。
   列表中带有“(default)”标记的配置项为当前默认配置项，如需进行变更，可选中所需配置项，并单击 ✓ 进行切换。

图 7-16 编译配置

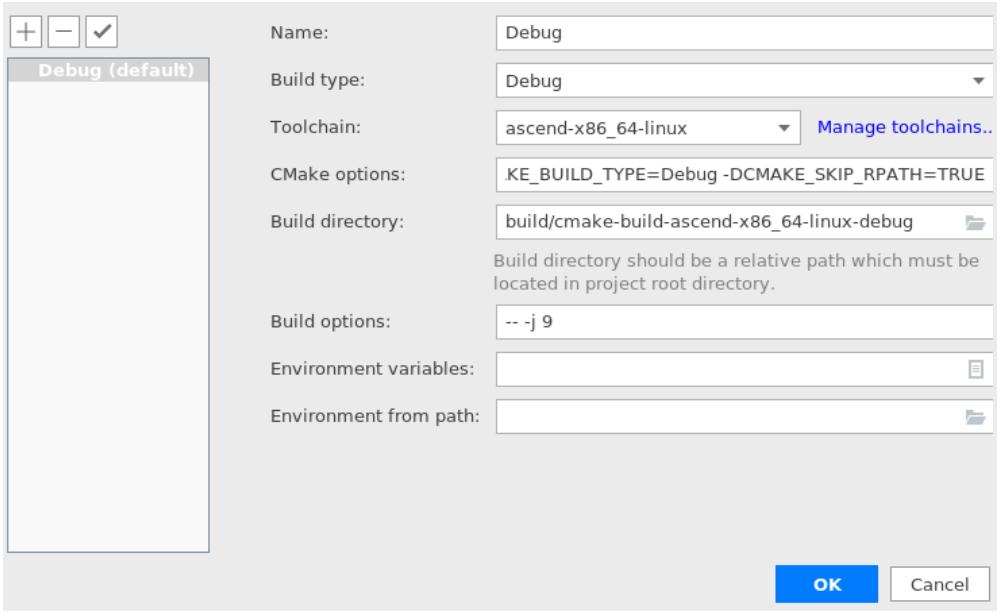


表 7-4 编译配置参数说明

参数配置	参数说明
Name	编译配置的名称。
Build Type	配置类型，可选：“Debug”或“Release”。
Toolchain	工具链配置器，配置详情请参见13.4 Toolchains，支持本地和远程编译功能。
CMake options	CMake选项，默认：“-DCMAKE_BUILD_TYPE=Debug -DCMAKE_SKIP_RPATH=TRUE”。
Build directory	编译目录相对路径，该路径是相对于工程目录的路径。
Build options	编译加速选项。
Environment Variables	环境变量配置。 可直接手动配置或单击 符号，在弹出窗中配置管理。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

2. 单击“OK”保存编译工程。

步骤4 请用户在MindStudio工程界面，按照使用场景选择以下编译方式，如需切换默认配置项请参见步骤3。

- 依次选择“Build > Clean CMake Project”，清理编译。（Clean CMake Project功能仅支持本地编译。）
- 依次选择“Build > Rebuild CMake Project”，使用默认配置进行全量编译。
- 依次选择“Build > Build CMake Project”或通过单击工具栏中的，使用默认配置进行增量编译。

----结束

运行应用工程（本地）

步骤1 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤2 单击左上角的“+”，选择“CMake Application”工程运行配置项，配置应用工程运行参数，图7-17为配置示例。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。

图 7-17 运行参数配置

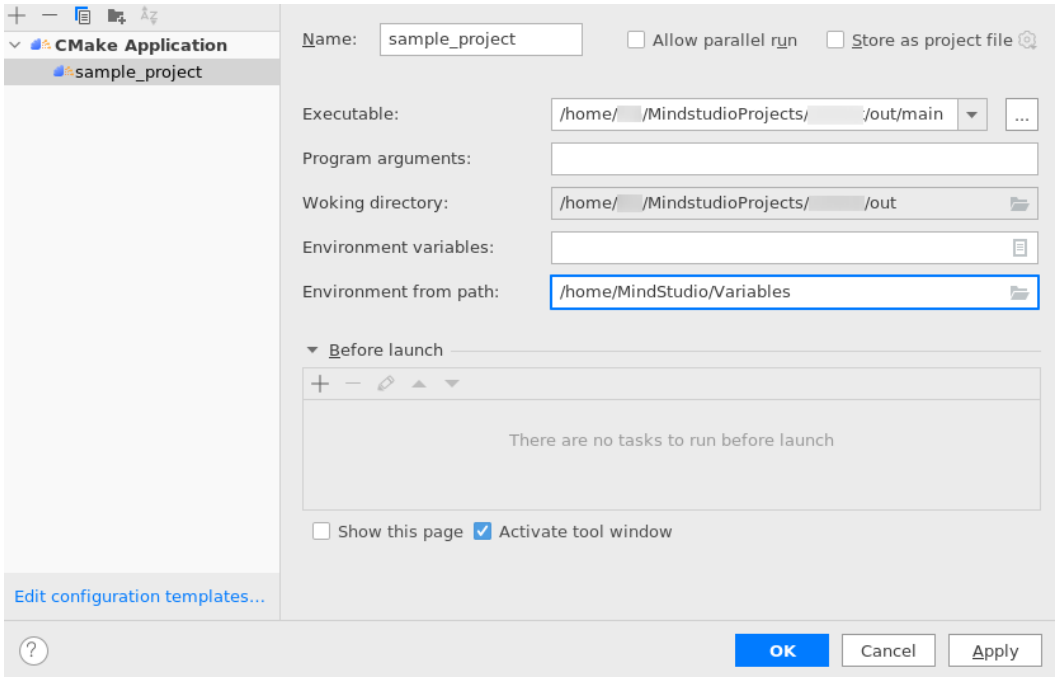


表 7-5 运行配置项

配置项	说明
Executable	选择编译用可执行文件的路径，选择到文件一级。
Program arguments	运行参数。
Working directory	工作目录。（如工程中配置了相对路径，则需在该选项中选择相对的工作目录。默认为可执行文件所在目录。）
Enviroment variables	此处直接输入动态链接库路径。 或者单击，在弹出的界面内单击，填写路径。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

步骤3 在MindStudio工程界面，顶部菜单栏中单击“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。
- 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《**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

运行应用工程（远端）

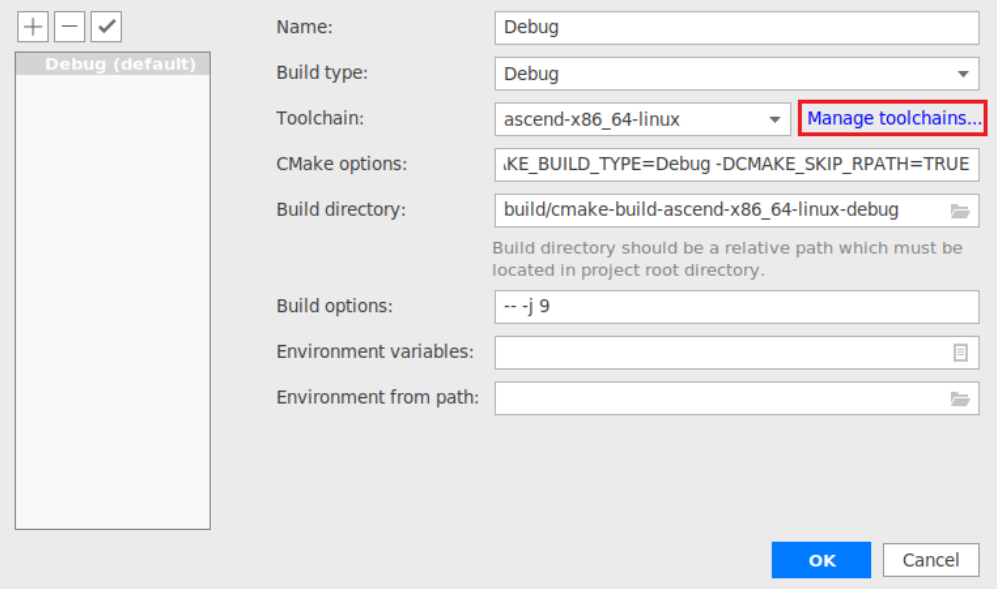
说明

- 远端运行时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“profiling”目录不会被拷贝。
- 对于Python工程，需添加“SSH Interpreter”解析器并在运行配置中，选择远端环境解析器即可远端运行，具体功能配置请参见13.9 Python SDK设置。

步骤1 配置远端Toolchain。

1. 在MindStudio工程界面，顶部菜单栏中单击“Build > Edit Configuration...”，进入编译配置页面。
2. 如图7-18所示，在“Toolchain”功能栏选择右侧“Manage toolchains...”，进入Toolchain功能配置。

图 7-18 选择 Manage toolchains



3. 配置远端环境Toolchain，具体操作请参见远程编译配置。
4. 在“Build Configuration”中选择配置完成的远端Toolchain，单击“OK”保存，单击工具栏中debug图标即可启动调试功能，具体调试功能请参考2.3.1.4 调试执行）。

步骤2 在左侧工程目录中，右键单击工程根目录，选择“Deployment > Upload to...”，选择并上传至目标远端环境。

步骤3 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤4 单击左上角按钮，新建“CMake Application”运行配置项并在“Executable”功能中选择远端可执行文件，点击“OK”完成配置。

步骤5 在MindStudio工程界面，顶部菜单栏中单击“Run > Run...”。

在弹出框中选择已创建好的运行配置信息，运行应用。

----结束

7.2.7.2 Windows 场景编译运行

须知

如果Windows本地时间晚于远程编译环境的时间，需要调整Windows本地时间与远程编译环境的时间，使两者保持一致后，通过工具栏“Build >Build CMake Project”方式执行编译。

编译应用工程

说明

- 若新建的工程为Python版本的应用工程，由于不需要执行“编译应用工程”，在MindStudio工程界面中“Build”功能不可用。
- 远端编译时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“model”、“profiling”目录不会被拷贝。
- 编译前，用户需确认工程目录下“set_env.cmake”文件的环境变量，替换其中的CANN软件包安装路径和MindSDK软件包安装路径。

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})
  ```

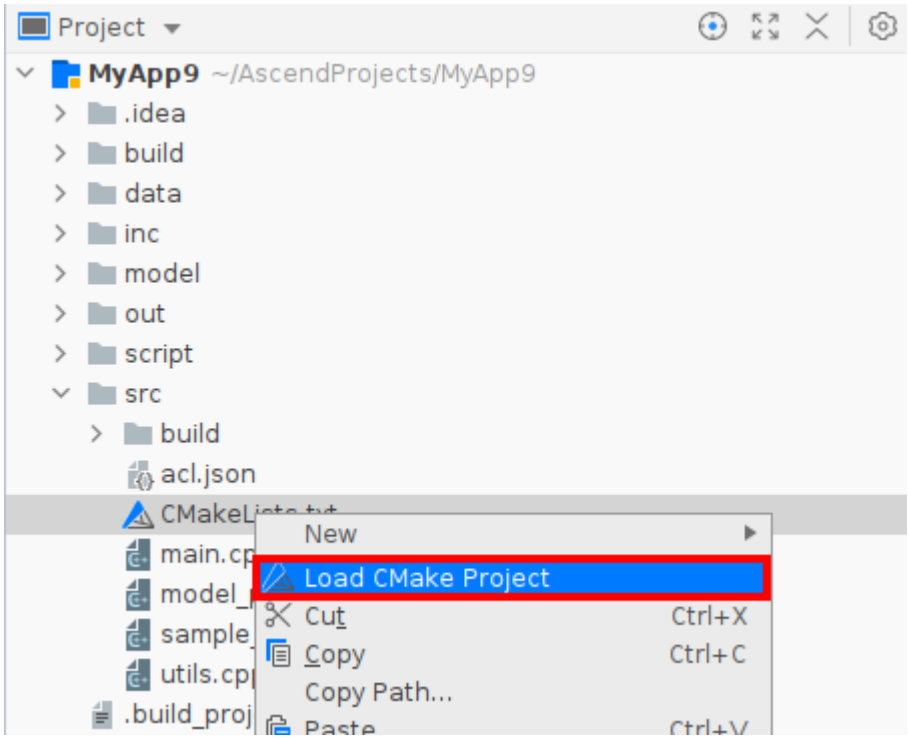
  说明

  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 （可选）指定“CMakeLists.txt”编译配置文件。

在工程界面左侧目录找到“CMakeLists.txt”文件，右键弹出并单击如图7-19所示“Load CMake Project”，即可指定此配置文件进行工程编译。

图 7-19 右键指定 CMakeLists.txt 编译配置文件

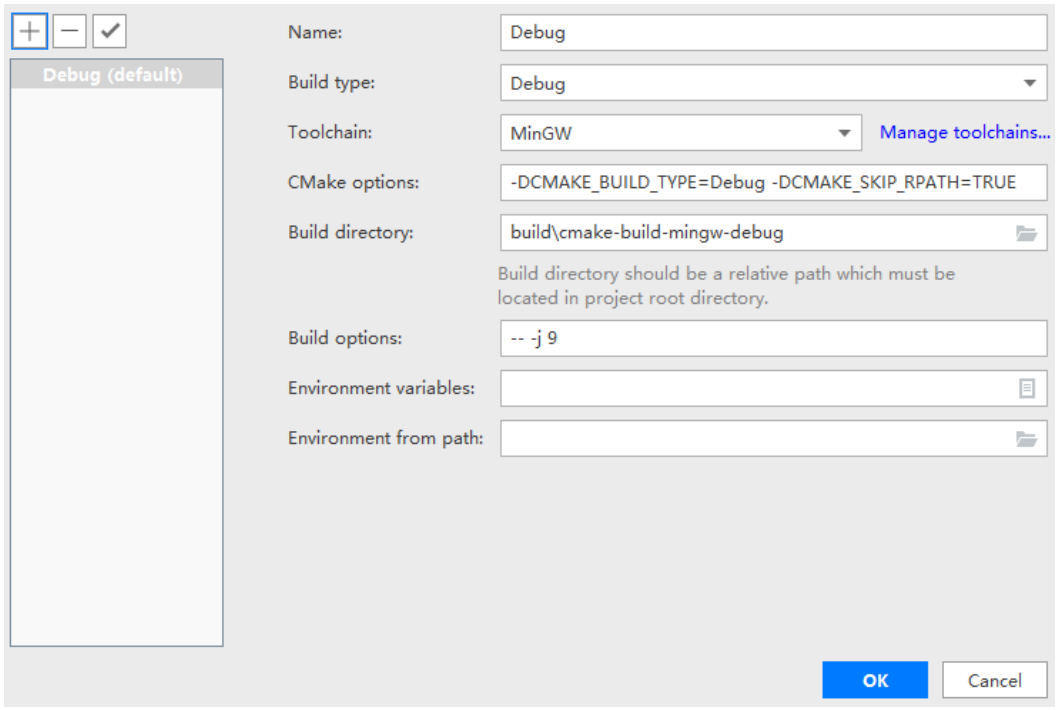


步骤3 编译配置。

在MindStudio工程界面，顶部菜单栏中单击“Build > Edit Configuration...”，进入编译配置页面（如7.2.7.2 Windows场景编译运行，图为配置示例）。

1. 单击 + 添加新增配置。
2. 列表中带有“(default)”标记的配置项为当前默认配置项，如需进行变更，可选中所需配置项，并单击 ✓ 进行切换（Toolchain工具链配置功能请参见13.4 Toolchains）。
3. 配置完成后单击“OK”保存编译配置。

图 7-20 编译配置



步骤4 请用户在MindStudio工程界面，按照使用场景选择以下编译方式，如需切换默认配置项请参见**步骤3**。

- 依次选择“Build > Clean CMake Project”，清理编译。（Clean CMake Project功能仅支持本地编译。）
- 依次选择“Build > Rebuild CMake Project”，使用默认配置进行增量编译。
- 依次选择“Build > Build CMake Project”通过单击工具栏中的  ，使用默认配置进行增量编译。

----结束

运行应用工程

说明

- 远端运行时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“profiling”目录不会被拷贝。
- 对于Python工程，需添加“SSH Interpreter”解析器并在运行配置中，选择远端环境解析器即可远端运行，具体功能配置请参见**13.9 Python SDK设置**。

步骤1 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤2 单击左上角“+”，选择“CMake Application”，新建配置应用工程运行参数，**图7-21**为配置示例。

图 7-21 运行参数配置

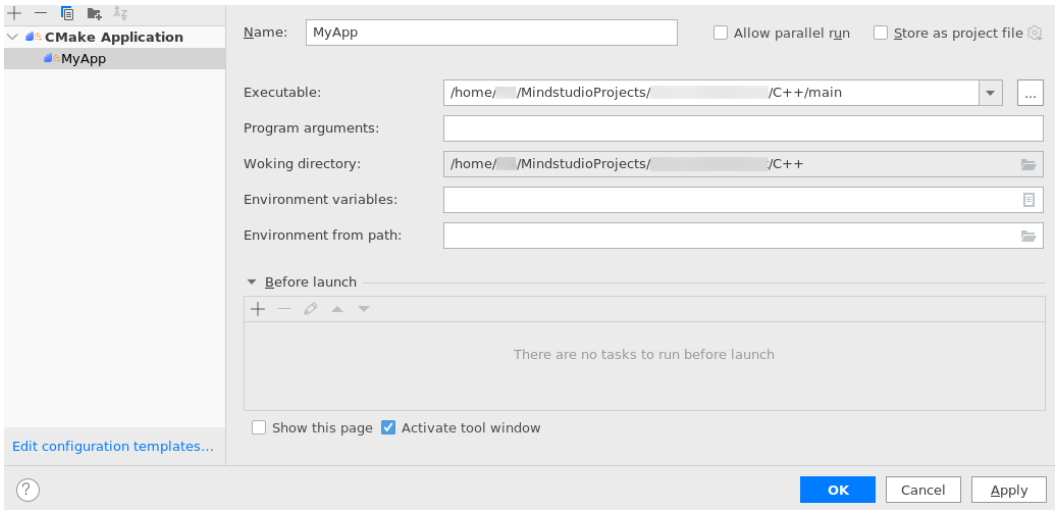


表 7-6 运行配置项

配置项	说明
Executable	选择编译用可执行文件的路径，选择到文件一级。
Program arguments	运行参数。
Working directory	工作目录。（如工程中配置了相对路径，则需在该选项中选择相对的工作目录。默认为可执行文件所在目录。）
Enviroment variables	此处直接输入动态链接库路径。 或者单击，在弹出的界面内单击，填写路径。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

步骤3 在MindStudio工程界面，顶部菜单栏中单击“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。
- 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《**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

7.2.8 SDK 样例工程使用指导

MindStudio提供基于MindSDK开发的样例工程，用户可以参考以下步骤创建样例工程。

图 7-22 MindSDK 样例工程创建流程

各步骤说明如下：

1. 环境准备：请参照**7.2.1 开发前须知**与**7.2.2 安装MindSDK软件包**章节，配置环境变量，安装MindSDK软件包。
2. 数据准备：请自行准备模型需要使用的数据集，如图片检测模型需要使用的图片。
3. 样例工程代码获取与导入：请参考**新建样例工程**从Gitee上获取所需的MindSDK Samples样例工程，并参考**导入应用工程**，将样例工程导入到MindStudio。
4. 获取模型：参考Gitee上样例工程的readme获取需要的模型并上传到环境中。更多模型可以参考**ModelZoo**。
5. 模型转换：请参考**6 模型转换和调优**将模型转换为om文件。
6. 应用编译与运行：请参考**7.2.7 编译与运行应用工程**对应用进行编译和运行。

7.3 基于 AscendCL 开发应用

7.3.1 开发前须知

- **对于运行环境在远端的Python版本应用工程**，在应用开发开始前需要对运行环境进行配置，其中*{version}*内容请根据实际情况填写。
  ```
  LD_LIBRARY_PATH=$HOME/Ascend/ascend-toolkit/{version}/runtime/lib64:$LD_LIBRARY_PATH;PYTHONPATH=$HOME/Ascend/ascend-toolkit/{version}/python/site-packages/acl:$PYTHONPATH
  ```
  可选择通过以下任意一种方式进行配置。
  - 修改运行环境中的“.bashrc”文件，加入“PYTHONPATH”与Runtime路径。
  - MindStudio编译/运行配置时，在“Run/Debug Configuration”功能中添加“Environment Variables”，补充环境变量，详细操作请参见**7.3.6 编译与运行应用工程**。
- 支持在Linux环境与Windows环境基于AscendCL开发应用。

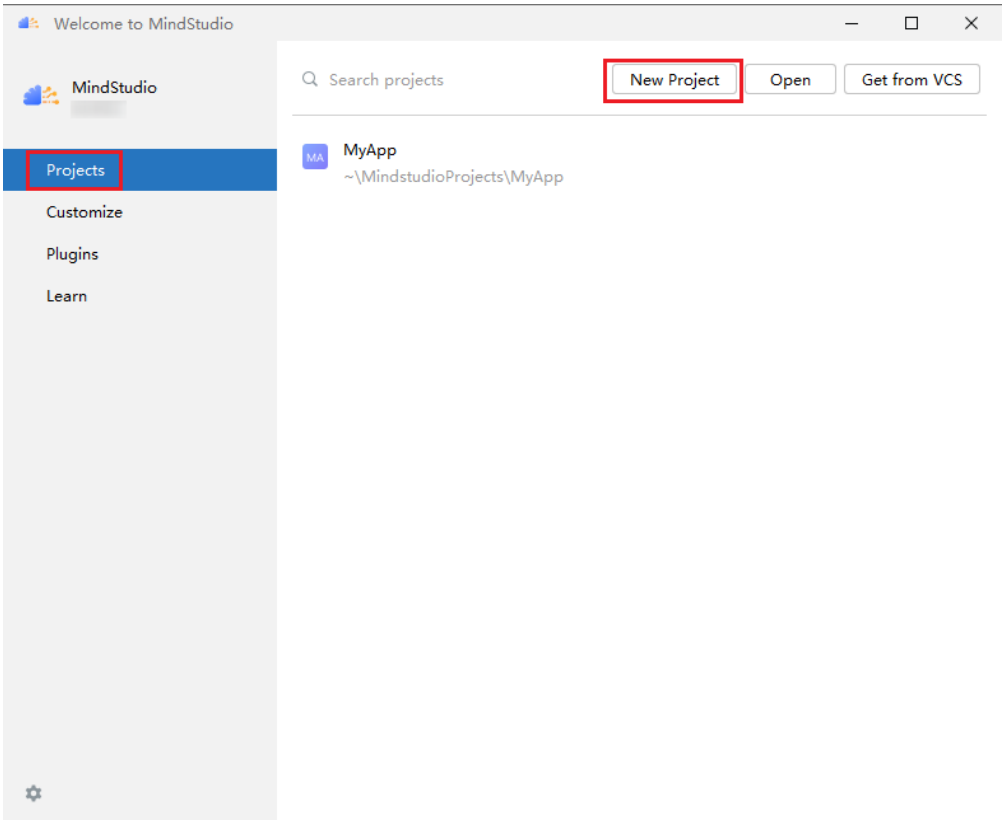
7.3.2 创建应用工程

新建模板工程

步骤1 进入工程创建页面。

- MindStudio欢迎界面：左侧菜单选择“Project”，右侧单击“New Project”。

图 7-23 工程创建界面



- MindStudio工程界面：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”，选择工程类型，如图7-24，单击“Next”进入工程配置。（若工程类型名称显示不全，鼠标悬停于工程类型名称处，则工程类型名称可以完整显示。）

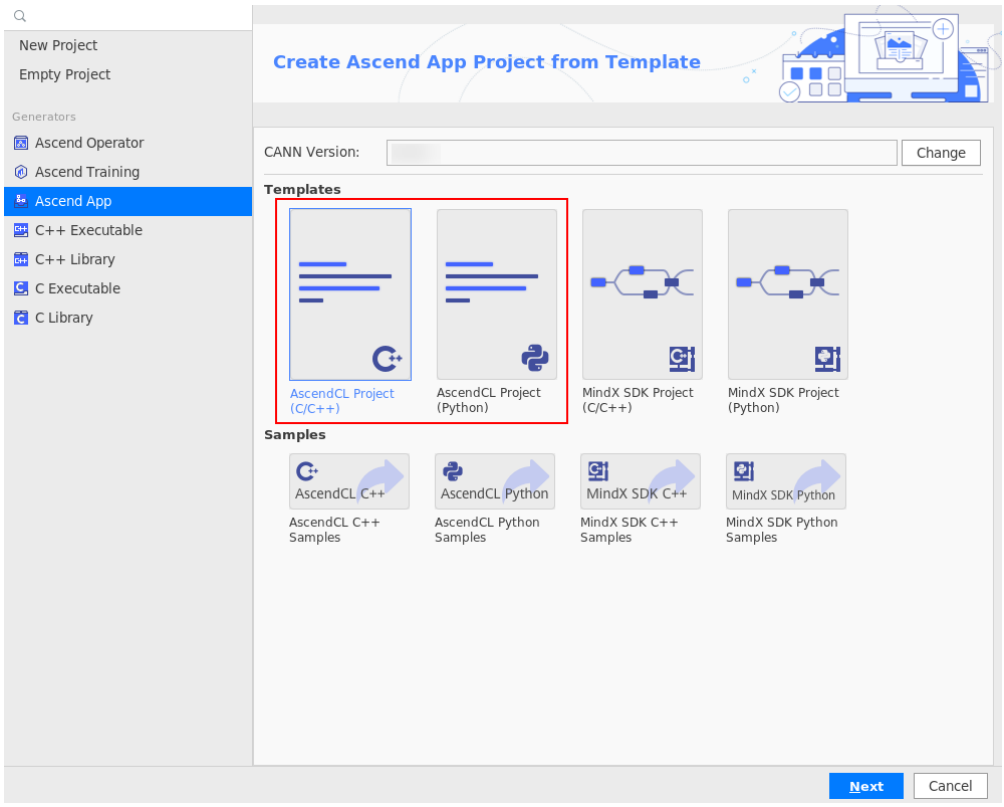
CANN Version：当前激活的CANN版本，可通过单击右侧“Change”进行变更，具体功能使用请参见13.6.2 切换/激活CANN包。

AscendCL Project为AscendCL空白工程，仅包括开发框架的工程，不含具体的代码逻辑。

说明

如未在MindStudio中配置CANN，部分应用工程可能无法正常显示。

图 7-24 AscendCL 应用工程



步骤3 进入工程配置栏，配置工程相关参数如表7-7所示。

表 7-7 工程参数说明

参数	说明
Project name	工程名称，自行配置。 名称开头和结尾必须是数字或字母。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。
Project location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）
More Settings	“Module name”：模块名，默认与“Project name”一致。
	“Content root”：根目录下路径。
	“Module file location”：模块文件路径。
	单击“Project format”右侧选框，出现下拉菜单。 • .idea（directory-based）：创建项目的时候创建一个.idea项来保存项目的信息，默认选项。 • .ipr（file-based）：项目配置文件来保存项目的配置信息。

步骤4 单击**Create**，完成工程创建。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。

- AscendCL Project(C/C++)：

```
├── 工程名
│   ├── build
│   │   ├──cmake            //存放cmake依赖文件。
│   ├── 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。
│   ├── CMakeLists.txt      //编译脚本，调用src目录下的CMakeLists文件。
```

- AscendCL Project(Python)：无代码目录。

----结束

导入应用工程

步骤1 导入工程文件，可通过选择以下任意一种方式完成。

- MindStudio欢迎界面：单击“Open”，选择需要导入的工程，单击“OK”确认导入。
- MindStudio工程界面：在顶部菜单栏中选择“File > Open...”或单击工具栏中的  ，选择现有工程打开。

  说明

  如该工程存在代码风险，在打开时会弹出信任窗口。

  - 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
  - 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
  - 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤2 若工作窗口已打开其他工程，会出现确认提示。

- 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。

步骤3 成功导入工程后，工程目录以树状呈现，请以实际创建结果为准。

----结束

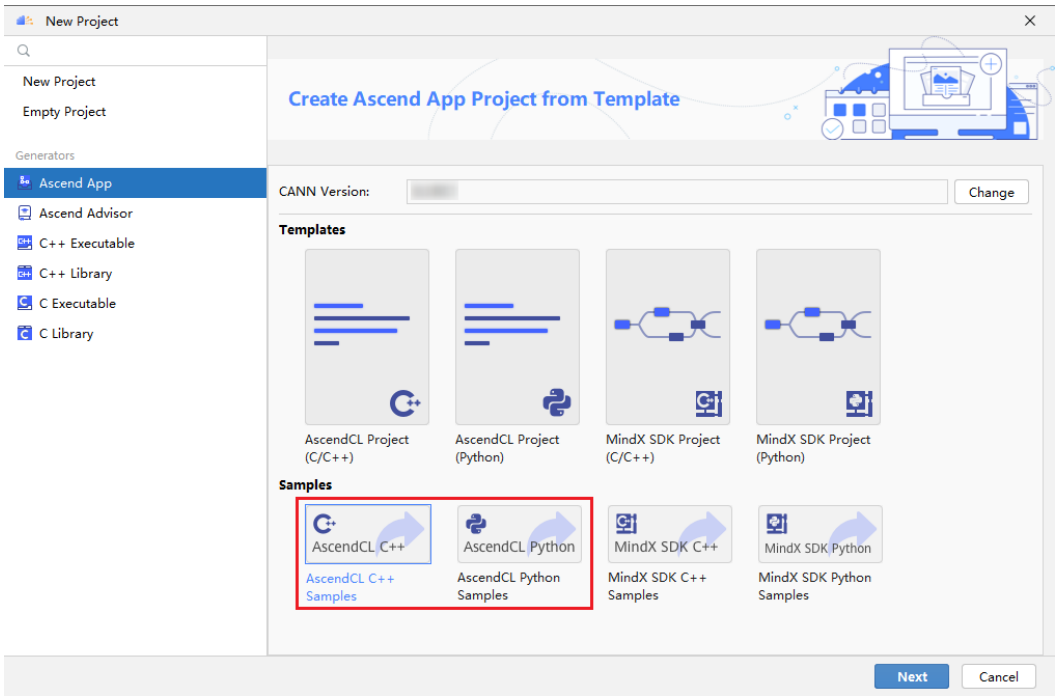
新建样例工程

步骤1 进入工程创建页面。

- MindStudio欢迎界面：单击“New Project”。
- MindStudio工程界面：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”，选择工程类型，如图7-25。（若工程类型名称显示不全，鼠标悬停于工程类型名称处，则工程类型名称可以完整显示。）

图 7-25 AscendCL Samples 样例工程



CANN Version：当前激活的CANN版本，可通过单击右侧“Change”进行变更，具体功能使用请参见13.6.2 切换/激活CANN包。

AscendCL C++ Samples和AscendCL Python Samples为基于AscendCL开发的样例工程。

说明

如未在MindStudio中配置CANN，部分应用工程可能无法正常显示。

步骤3 单击“Next”，浏览器会跳转至对应的Gitee代码仓界面。

步骤4 在Gitee代码仓页面下单击“克隆/下载 > 复制”，复制代码包下载链接。

步骤5 在开发环境执行命令：git clone URL（其中URL为复制的代码包下载链接），直接将代码包克隆到开发环境。

```
git clone https://gitee.com/ascend/samples.git
```

步骤6 在下载的文件夹中选择需要的样例，然后直接通过MindStudio导入，详情请参见导入应用工程。

----结束

7.3.3 准备模型文件和数据

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6 模型转换和调优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。

请用户自行将准备好的模型om文件上传到创建的工程中。

步骤3 （可选）如果需要使用dump功能：

1. 准备xxx.json（例如acl.json）配置文件。

   请参见**9.5.3.4 准备离线模型dump数据**准备。

2. 通过以下任意一种方式将配置文件作为输入参数完成dump配置。
   - 在aclInit接口（acl.init接口）中将配置文件作为输入参数完成dump配置。
   - 在aclInit接口（acl.init接口）之后、模型加载接口之前调用aclmdlSetDump接口（acl.mdl.set_dump接口）。

   具体配置方式请参见《**AscendCL应用软件开发指南（C&C++）**》或《**AscendCL应用软件开发指南（Python）**》。

说明

当前仅支持本地dump configuration配置，若需远端dump，需用户自行配置acl.json。

步骤4 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

7.3.4 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，提高用户开发效率。

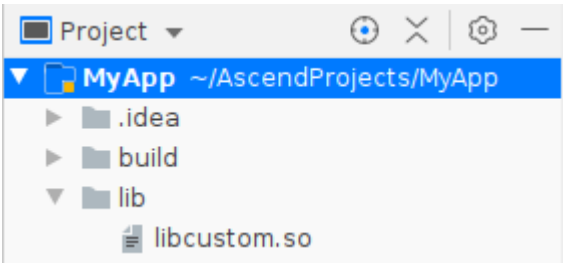
用户添加自定义动态链接库的步骤参考如下：

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7-26**中lib目录下的libcustom.so）。

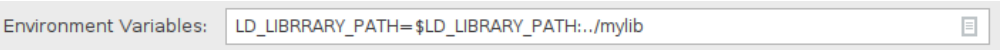
图 7-26 存放动态链接库的目录



新建的存放动态链接库的目录，需要在应用工程运行参数配置（**7.3.6 编译与运行应用工程**）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所在位置“*工程名*/out/main”为基准，即截图中配置的“../mylib”位置，“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：

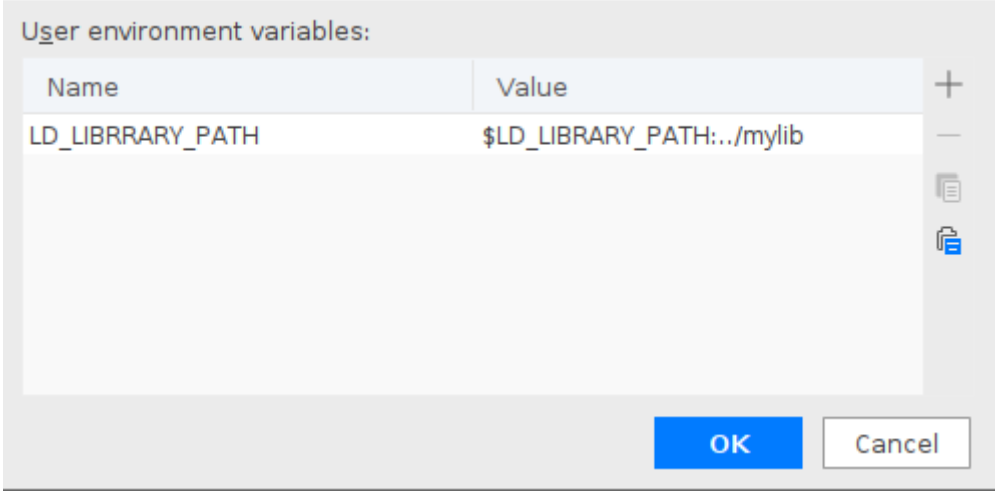
- 在Environment Variables处直接输入，如图图7-27。

  图 7-27 动态链接库目录

- 单击后在出的弹窗内单击，填写。

  图 7-28 环境变量填写

----结束

7.3.5 开发应用

创建 AscendCL Project 时

请参见《AscendCL应用软件开发指南（C&C++）》或《AscendCL应用软件开发指南（Python）》进行应用开发。

创建 AscendCL Samples 样例工程时

- 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（以C++工程为例）。

    ```
    //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（以C/C++工程为例）。

    ```
    //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，请参见《AscendCL应用软件开发指南（C&C++）》或《AscendCL应用软件开发指南（Python）》。

- 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《**AscendCL应用软件开发指南（C&C++）**》或《**AscendCL应用软件开发指南（Python）**》开发应用。

7.3.6 编译与运行应用工程

7.3.6.1 Linux 场景编译运行

编译应用工程

说明

- 若新建的工程为Python版本的应用工程，由于不需要执行“编译应用工程”，在MindStudio工程界面中“Build”功能不可用。
- 远端编译时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“model”、“profiling”目录不会被拷贝。
- 编译前，用户需确认工程目录下“set_env.cmake”文件的环境变量，替换其中的CANN软件包安装路径。

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})
  ```

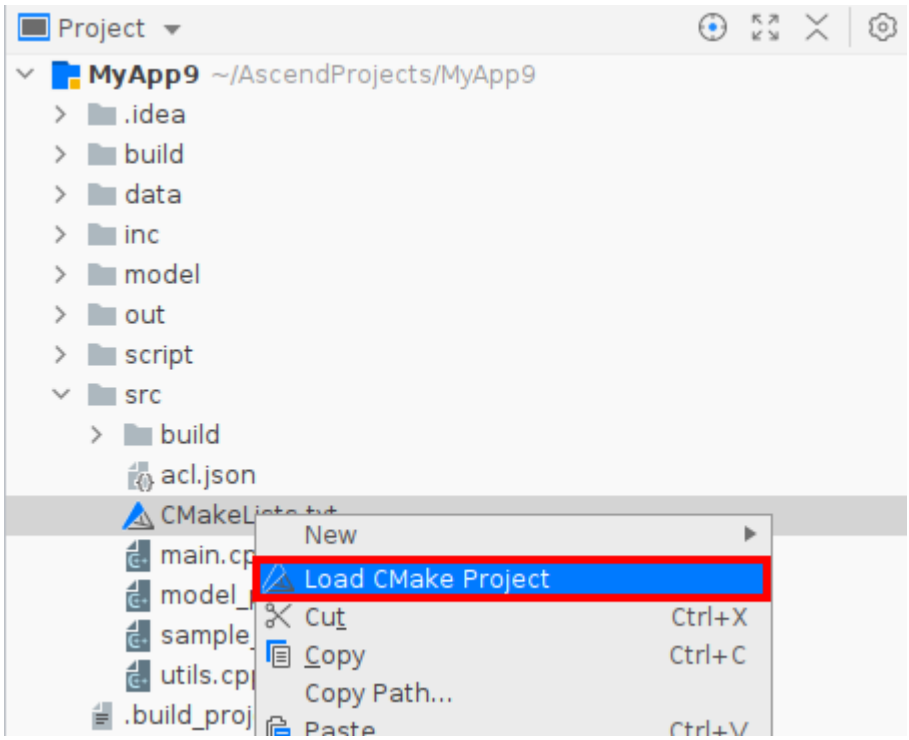
说明

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 （可选）指定“CMakeLists.txt”编译配置文件。

在工程界面左侧目录找到“CMakeLists.txt”文件，右键弹出并单击如图7-29所示“Load CMake Project”，即可指定此配置文件进行工程编译。

图 7-29 右键指定 CMakeLists.txt 编译配置文件



步骤3 编译配置。

在MindStudio工程界面，顶部菜单栏单击“Build > Edit Configuration...”，进入编译配置页面。

1. 配置编译参数，单击 + 添加新增配置。
   列表中带有“(default)”标记的配置项为当前默认配置项，如需进行变更，可选中所需配置项，并单击 ✓ 进行切换。

图 7-30 编译配置

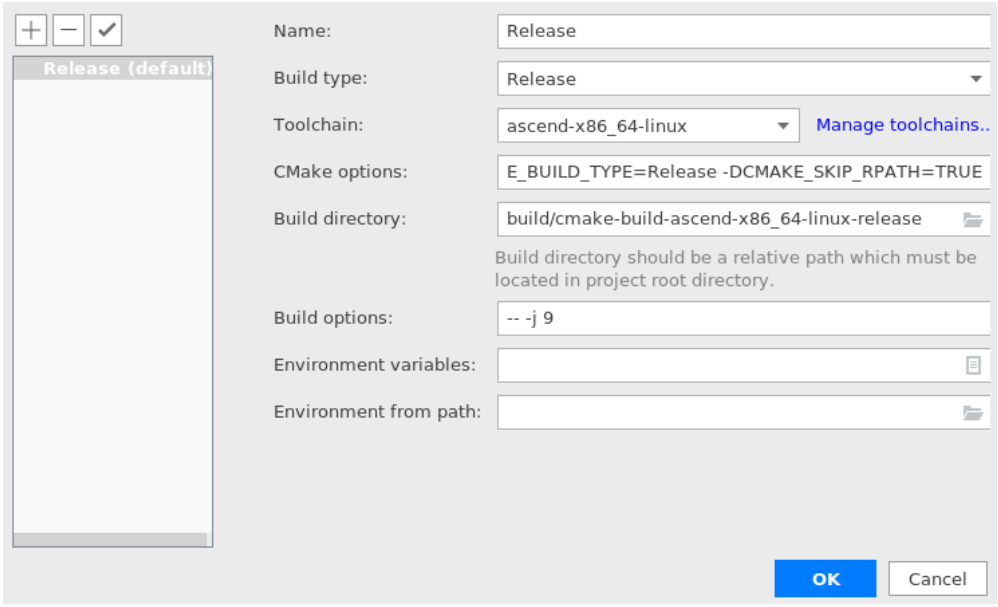


表 7-8 编译配置参数说明

参数配置	参数说明
Name	编译配置的名称。
Build Type	配置类型，可选：“Debug”或“Release”。
Toolchain	工具链配置器，配置详情请参见13.4 Toolchains，支持本地和远程编译功能。
CMake options	CMake选项，默认：“-DCMAKE_BUILD_TYPE=Debug -DCMAKE_SKIP_RPATH=TRUE”。
Build directory	编译目录相对路径，该路径是相对于工程目录的路径。
Build options	编译加速选项。
Environment Variables	环境变量配置。 可直接手动配置或单击 符号，在弹出窗中配置管理。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

2. 单击“OK”保存编译工程。

步骤4 请用户在MindStudio工程界面，按照使用场景选择以下编译方式。

- 依次选择“Build > Clean CMake Project”，清理编译。（Clean CMake Project功能仅支持本地编译。）
- 依次选择“Build > Rebuild CMake Project”，使用默认配置进行全量编译。
- 依次选择“Build > Build CMake Project”或通过单击工具栏中的  ，使用默认配置进行增量编译。

提示“Build finished”，则表示编译完成。

----结束

运行应用工程（本地）

步骤1 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤2 单击左上角的“+”，选择“CMake Application”工程运行配置项，配置应用工程运行参数，图7-31为配置示例。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。

图 7-31 运行参数配置

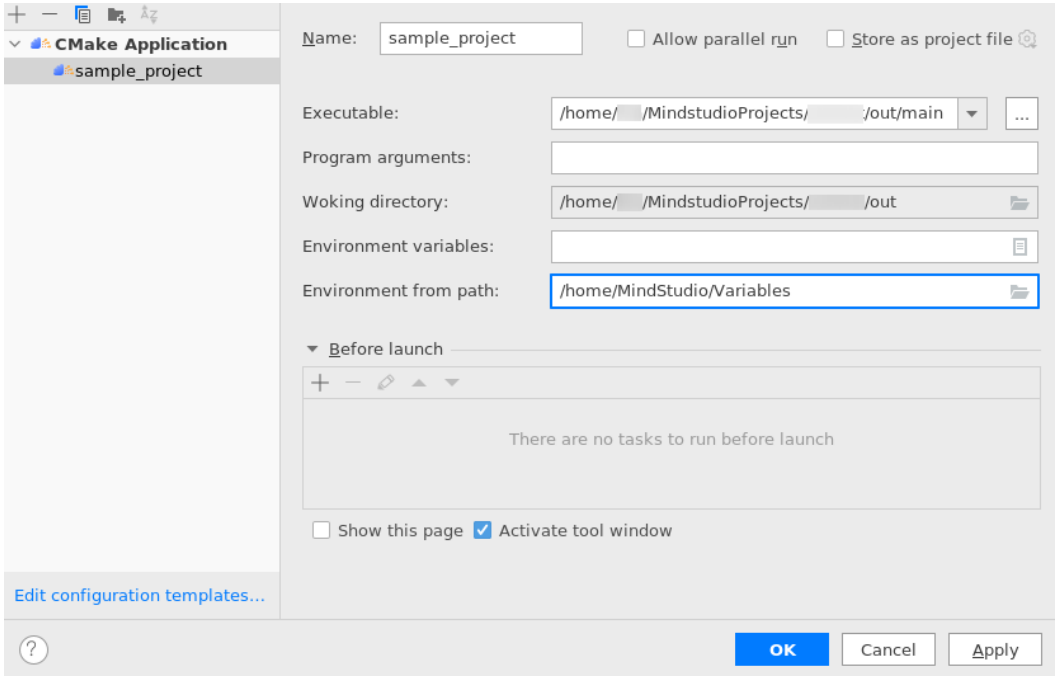


表 7-9 运行配置项

配置项	说明
Executable	选择编译用可执行文件的路径，选择到文件一级。
Program arguments	运行参数。
Working directory	工作目录。（如工程中配置了相对路径，则需在该选项中选择相对的工作目录。默认为可执行文件所在目录。）

配置项	说明
Enviroment variables	此处直接输入动态链接库路径。 或者单击，在弹出的界面内单击，填写路径。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

步骤3 在MindStudio工程界面，顶部菜单栏中单击“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。
- 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《**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

运行应用工程（远端）

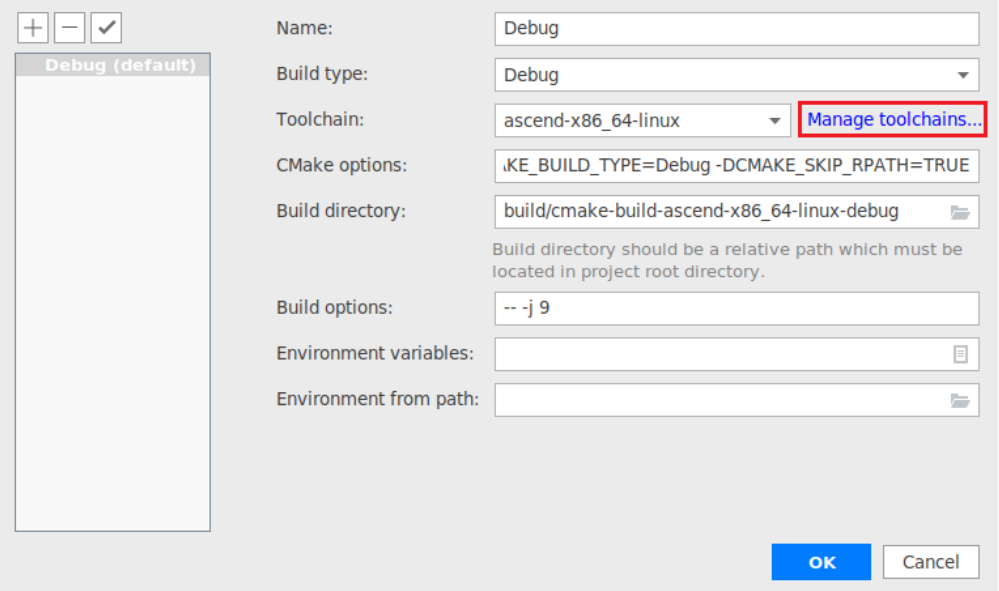
说明

- 远端运行时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“profiling”目录不会被拷贝。
- 对于Python工程，需添加“SSH Interpreter”解析器并在运行配置中，选择远端环境解析器即可远端运行，具体功能配置请参见**13.9 Python SDK设置**。

步骤1 配置远端Toolchain。

1. 在MindStudio工程界面，顶部菜单栏中单击“Build > Edit Configuration...”，进入编译配置页面。
2. 如**图7-32**所示，在“Toolchain”功能栏选择右侧“Manage toolchains...”，进入Toolchain功能配置。

图 7-32 选择 Manage toolchains



3. 配置远端环境Toolchain，具体操作请参见**远程编译配置**。
4. 在“Build Configuration”中选择配置完成的远端Toolchain，单击“OK”保存，单击工具栏中debug图标即可启动调试功能，具体调试功能请参考**2.3.1.4 调试执行**）。

步骤2 在左侧工程目录中，右键单击工程根目录，选择“Deployment > Upload to...”，选择并上传至目标远端环境。

步骤3 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤4 单击左上角按钮，新建“CMake Application”运行配置项并在“Executable”功能中选择远端可执行文件，点击“OK”完成配置。

步骤5 在MindStudio工程界面，顶部菜单栏中单击“Run > Run...”。

在弹出框中选择已创建好的运行配置信息，运行应用。

----结束

7.3.6.2 Windows 场景编译运行

须知

如果Windows本地时间晚于远程编译环境的时间，需要调整Windows本地时间与远程编译环境的时间，使两者保持一致后，通过工具栏“Build >Build CMake Project”方式执行编译。

编译应用工程

说明

- 若新建的工程为Python版本的应用工程，由于不需要执行“编译应用工程”，在MindStudio工程界面中“Build”功能不可用。
- 远端编译时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“model”、“profiling”目录不会被拷贝。
- 编译前，用户需确认工程目录下“set_env.cmake”文件的环境变量，替换其中的CANN软件包安装路径。

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})
  ```

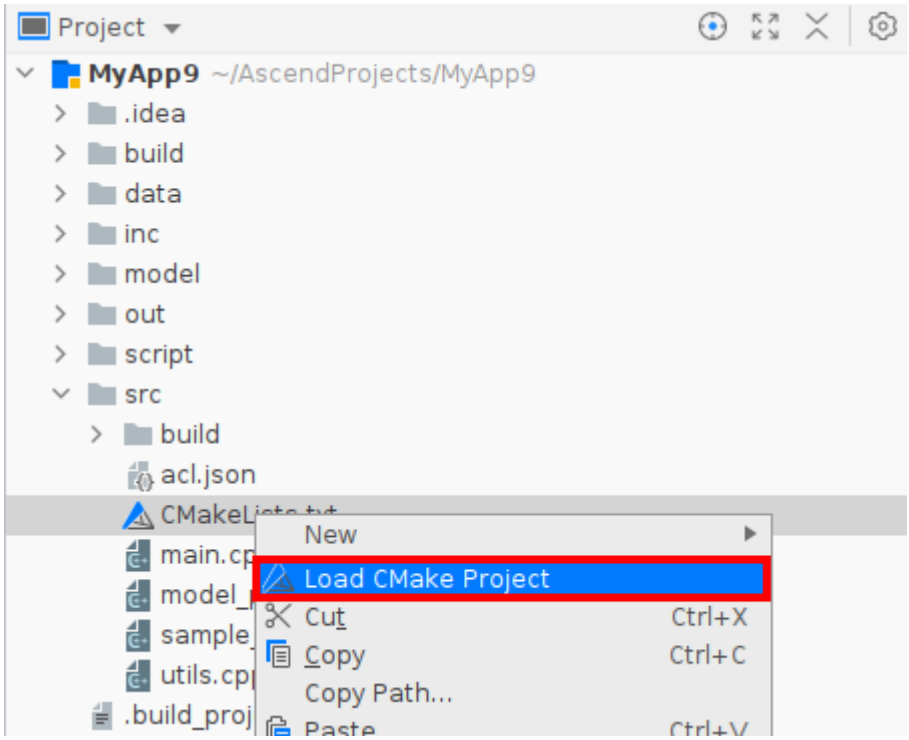
说明

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 （可选）指定“CMakeLists.txt”编译配置文件。

在工程界面左侧目录找到“CMakeLists.txt”文件，右键弹出并单击如**图7-33**所示“Load CMake Project”，即可指定此配置文件进行工程编译。

图 7-33 右键指定 CMakeLists.txt 编译配置文件

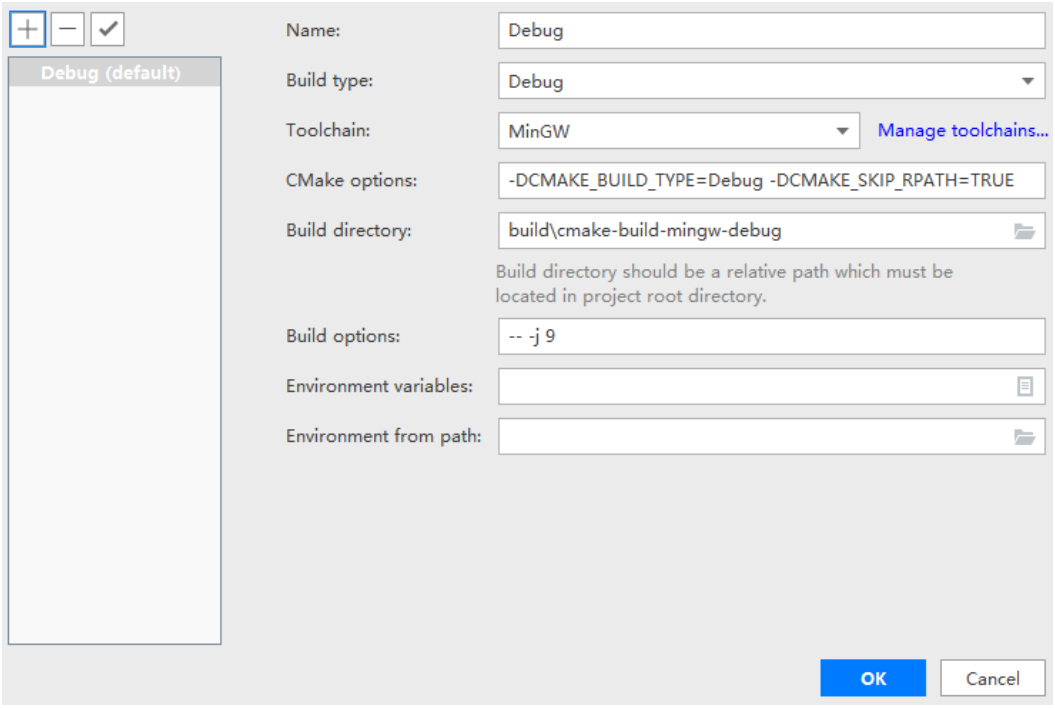


步骤3 编译配置。

在MindStudio工程界面，顶部菜单栏中单击“Build > Edit Configuration...”，进入编译配置页面（如**7.2.7.2 Windows场景编译运行**，图为配置示例）。

1. 单击 + 添加新增配置。
2. 列表中带有“(default)”标记的配置项为当前默认配置项，如需进行变更，可选中所需配置项，并单击 ✓ 进行切换（Toolchain工具链配置功能请参见**13.4 Toolchains**）。
3. 配置完成后单击“OK”保存编译配置。

图 7-34 编译配置



步骤4 请用户按照使用场景选择以下编译方式。

- 在MindStudio工程界面，依次选择“Build > Clean CMake Project”，清理编译。（Clean CMake Project功能仅支持本地编译。）
- 在MindStudio工程界面，依次选择“Build > Rebuild CMake Project”，使用默认配置进行全量编译。
- 在MindStudio工程界面，依次选择“Build > Build CMake Project”或通过单击工具栏中的，使用默认配置进行增量编译。

----结束

运行应用工程

说明

- 远端运行时，会对工程文件夹进行目录拷贝到远端连接的环境，工程文件夹中“dump”、“profiling”目录不会被拷贝。
- 对于Python工程，需添加“SSH Interpreter”解析器并在运行配置中，选择远端环境解析器即可远端运行，具体功能配置请参见13.9 Python SDK设置。

步骤1 在MindStudio工程界面，顶部菜单栏中单击“Run > Edit Configurations...”，进入运行配置页面。

步骤2 单击左上角“+”，选择“CMake Application”，新建配置应用工程运行参数，图7-35为配置示例。

图 7-35 运行参数配置

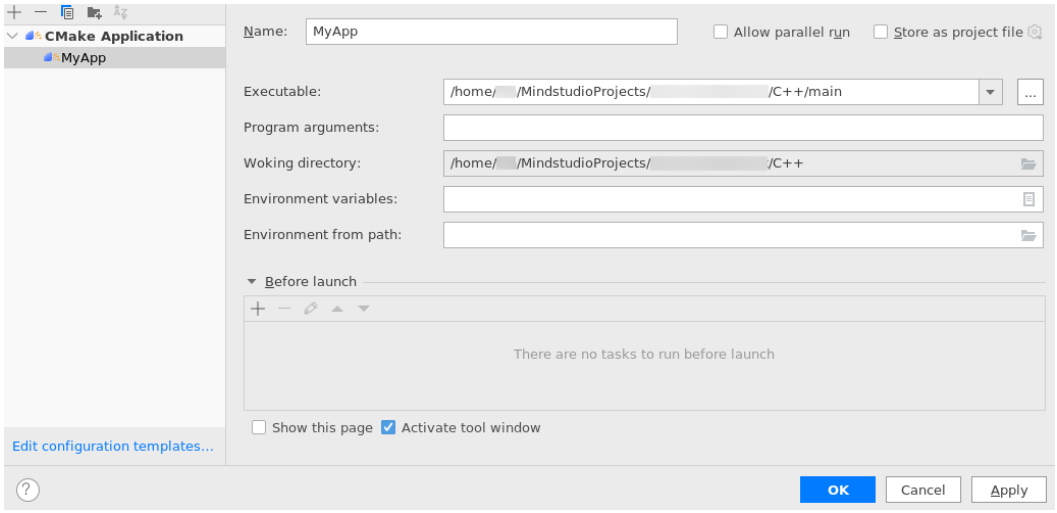


表 7-10 运行配置项

配置项	说明
Executable	选择编译用可执行文件的路径，选择到文件一级。
Program arguments	运行参数。
Working directory	工作目录。（如工程中配置了相对路径，则需在该选项中选择相对的工作目录。默认为可执行文件所在目录。）
Enviroment variables	此处直接输入动态链接库路径。 或者单击，在弹出的界面内单击，填写路径。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

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，顶部菜单栏中单击“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。
- 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《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

7.3.7 AscendCL 样例工程使用指导

MindStudio提供基于AscendCL开发的样例工程，用户可以参考以下步骤创建样例工程。

图 7-36 AscendCL 样例工程创建流程

各步骤说明如下：

1. 环境准备：请参照**7.3.1 开发前须知**配置环境变量。
2. 数据准备：请自行准备模型需要使用的数据集，如图片检测模型需要使用的图片。
3. 样例工程代码获取与导入：请参考**新建样例工程**从Gitee上获取所需的AscendCL样例工程，并参考**导入应用工程**导入到MindStudio。
4. 获取模型：参考Gitee上样例工程的readme获取需要的模型并上传到环境中。更多模型可以参考**ModelZoo**。
5. 模型转换：请参考**6 模型转换和调优**将模型转换为om文件。
6. 应用编译与运行：请参考**7.3.6 编译与运行应用工程**对应用进行编译和运行。

8 算子开发

8.1 简介

介绍如何使用MindStudio工具实现不同框架的算子开发。

包括支持MindSpore、PyTorch、TensorFlow、Caffe、ONNX的TBE算子开发和支持TensorFlow、PyTorch、ONNX、MindSpore的AI CPU算子开发，其中MindSpore框架的算子目前仅支持Atlas 训练系列产品。

本章节主要介绍**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，请参见对应《**TBE&AI CPU算子开发指南**》。

配置完成Ascend-cann-toolkit开发套件包后，可从***Ascend-cann-toolkit安装目录*/ascend-toolkit/latest**/tools/msopgen/template/operator_demo_projects/获取算子开发样例。样例请参见**8.12.1.1 导入算子工程样例**。

***Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/opp/built-in/op_impl/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 算子开发依赖

算子开发前，须确保MindStudio运行的Python环境安装以下依赖。

如下命令如果使用非root用户安装，需要在安装命令后加上--user，例如：**pip3 install xlrd==1.2.0 --user**。

表 8-1 算子开发 MindStudio 需要的 Python 依赖

依赖包	版本要求	应用场景	参考安装方法（以python3在X86架构为例）
xlrd	1.2.0	创建算子工程	pip3 install xlrd==1.2.0
gnureadline	-	TIK开发	pip3 install gnureadline
mindspore	-	MindSpore算子开发	安装方式参考**MindSpore安装指南**。
absl-py	-	UT测试	pip3 install absl-py
coverage	-		pip3 install coverage
jinja2	3.1.2		pip3 install jinja2
onnx	-	ST测试	pip3 install onnx
tensorflow	1.15.0		pip3 install tensorflow==1.15.0
csv	-		pip3 install python-csv
google	-		pip3 install google

说明

aarch64架构的操作系统，安装依赖时可能会安装失败。请使用源码编译安装或者参考对应依赖的官方安装指导操作。

8.3 开发流程

8.3.1 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下：

- TensorFlow/ONNX/Caffe TBE算子开发流程如**图8-1**所示。

- PyTorch TBE算子开发流程如图8-2所示。
- MindSpore TBE算子开发流程如图8-3所示。

图 8-1 TensorFlow/ONNX/Caffe TBE 算子开发流程

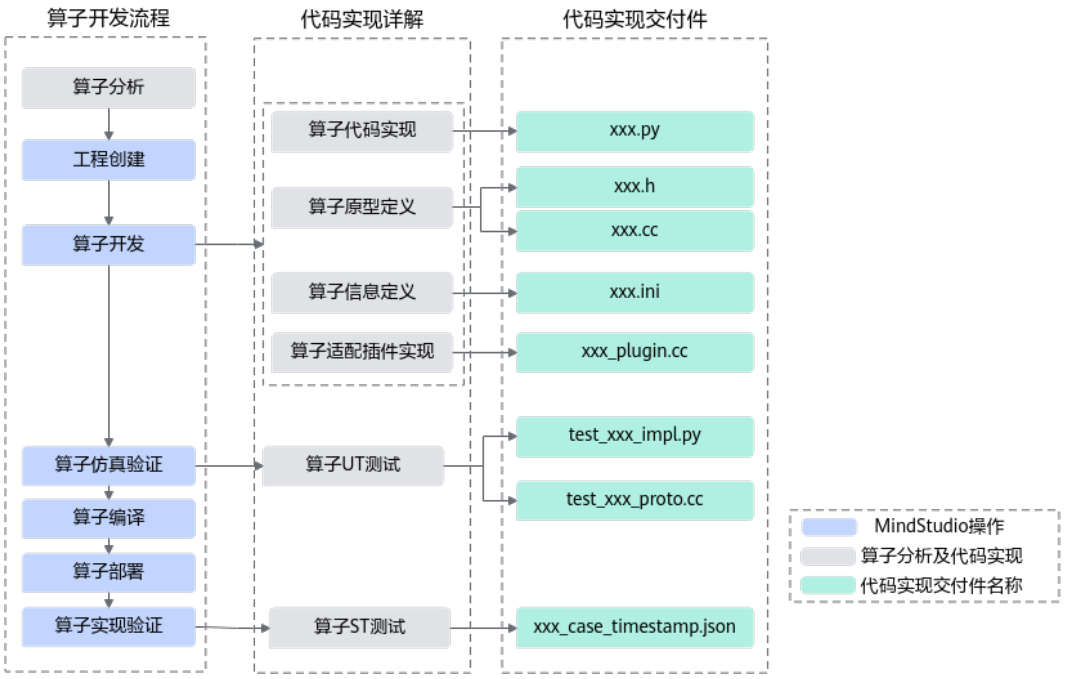


图 8-2 PyTorch TBE 算子开发流程

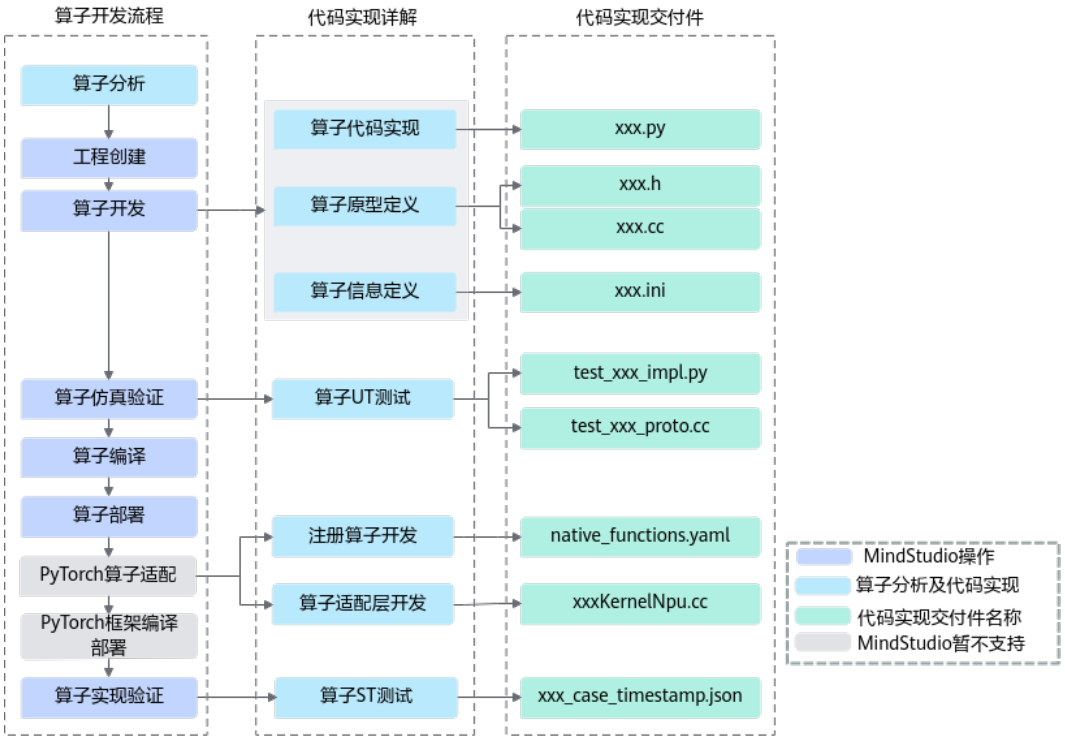
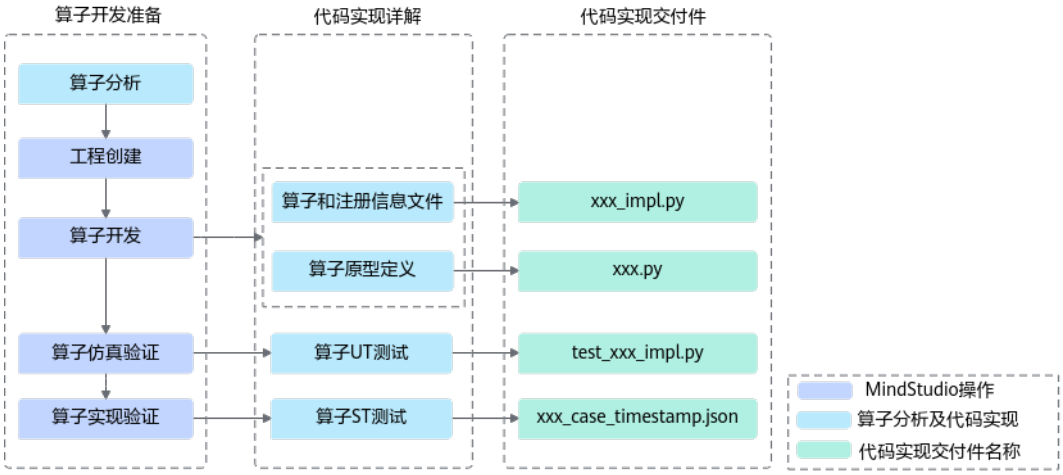


图 8-3 MindSpore TBE 算子开发流程



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. **创建算子工程**：通过MindStudio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**：MindSpore框架的算子实现和注册信息文件以及算子原型定义具体实现方式请参考《MindSpore教程》的“**自定义算子**”。
   - **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。
   - **算子信息库定义**：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/ONNX/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第三方框架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。基于第三方框架的网络运行时，首先会加载并调用GE中的插件信息，将第三方框架网络中的算子进行解析并映射成昇腾AI处理器中的算子。
4. **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。
5. **算子编译**：将算子插件实现文件编译成算子插件，算子原型定义文件编译成算子原型库，算子信息定义文件编译成算子信息库。
6. **算子部署**：将算子实现文件、编译后的算子插件、算子原型库和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。
7. **PyTorch适配**：昇腾AI处理器具有内存管理、设备管理和算子调用实现功能。PyTorch算子适配根据PyTorch原生结构进行昇腾AI处理器扩展。

8. **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实现代码的功能正确性。

8.3.2 AI CPU 算子开发流程

通过MindStudio工具进行AI CPU算子开发的总体开发流程如下：

- TensorFlow/ONNX/PyTorch AI CPU算子开发流程如图8-4所示。
- MindSpore AI CPU算子开发流程如图8-5所示。

图 8-4 TensorFlow/ONNX/PyTorch AI CPU 算子开发流程

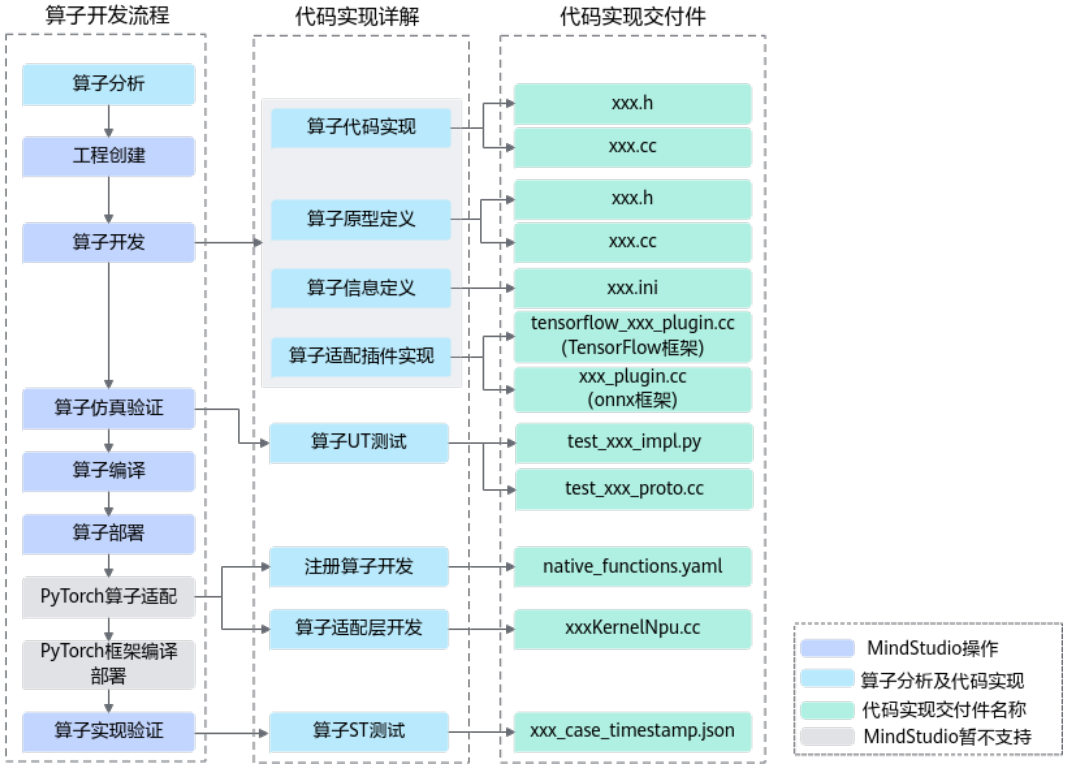
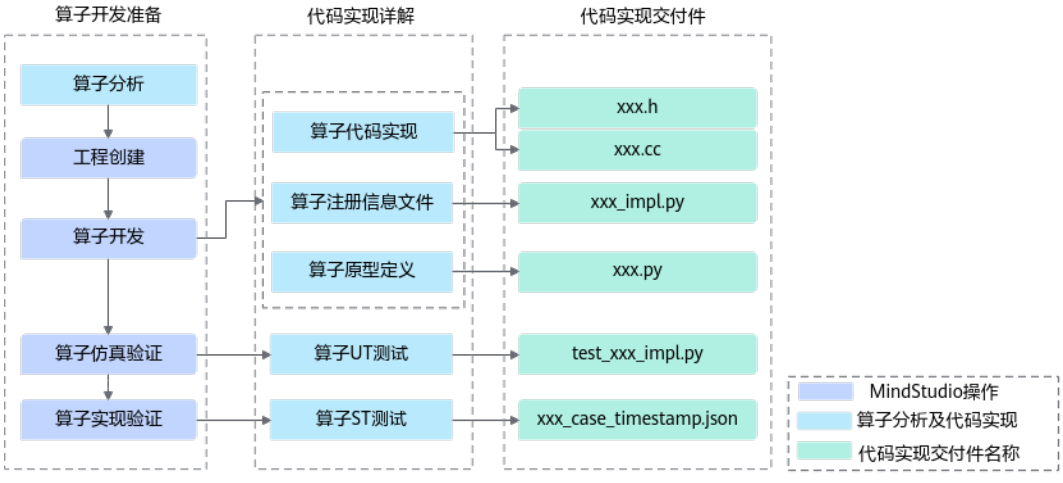


图 8-5 MindSpore AI CPU 算子开发流程



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. **创建算子工程**：通过MindStudio工具创建AI CPU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**：MindSpore框架的算子实现和注册信息文件以及算子原型定义具体实现方式请参考《MindSpore教程》的“**自定义算子**”。
   - **算子代码实现**：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。
   - **算子信息库定义**：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/ONNX）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/ONNX的算子映射成昇腾AI处理器的算子。
4. **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。
5. **算子编译**：将算子适配插件实现文件编译成算子插件，原型定义文件编译成算子原型库，信息定义文件编译成算子信息库。
6. **算子部署**：将算子实现文件、编译后的算子插件、算子原型库和算子信息库部署到昇腾AI处理器算子库中（opp的对应目录下）。
7. **PyTorch适配**：昇腾AI处理器具有内存管理，设备管理和算子调用实现功能。PyTorch算子适配根据PyTorch原生结构进行昇腾AI处理器扩展。
8. **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实现代码的功能正确性。

8.4 算子分析

简介

本节以Add TBE算子和AI CPU算子开发为例进行算子分析，帮助读者快速掌握算子分析流程。

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：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tbe.dsl.vadd(lhs, rhs)接口实现x+y，请参考《**TBE&AI CPU算子开发指南**》中的“算子开发过程 > 算子代码实现（TBE DSL）”章节。
2. 由于tbe.dsl.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tbe.dsl.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**8.6.4 算子信息库定义**中配置**opFile.value**与**opInterface.value**。
  - 不配置**8.6.4 算子信息库定义**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。

步骤5 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	tbe.dsl.broadcast(var, shape, output_dtype=None) tbe.dsl.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。

步骤4 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

8.5 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，开发者可以基于这些模板进行算子开发。

配置 Python SDK

算子开发之前，开发者需要参考**13.9 Python SDK设置**章节设置算子工程依赖Python库。

操作步骤

步骤1 进入算子工程创建界面。

- MindStudio欢迎界面：单击“New Project”，进入图8-6。
- MindStudio工程界面：
  - 在非算子工程中新建算子工程：在顶部菜单栏中选择“File > New > Project...”，进入图8-6。
  - 在算子工程中新建算子工程：在顶部工具栏中单击  ，进入图8-7。

图 8-6 创建工程界面

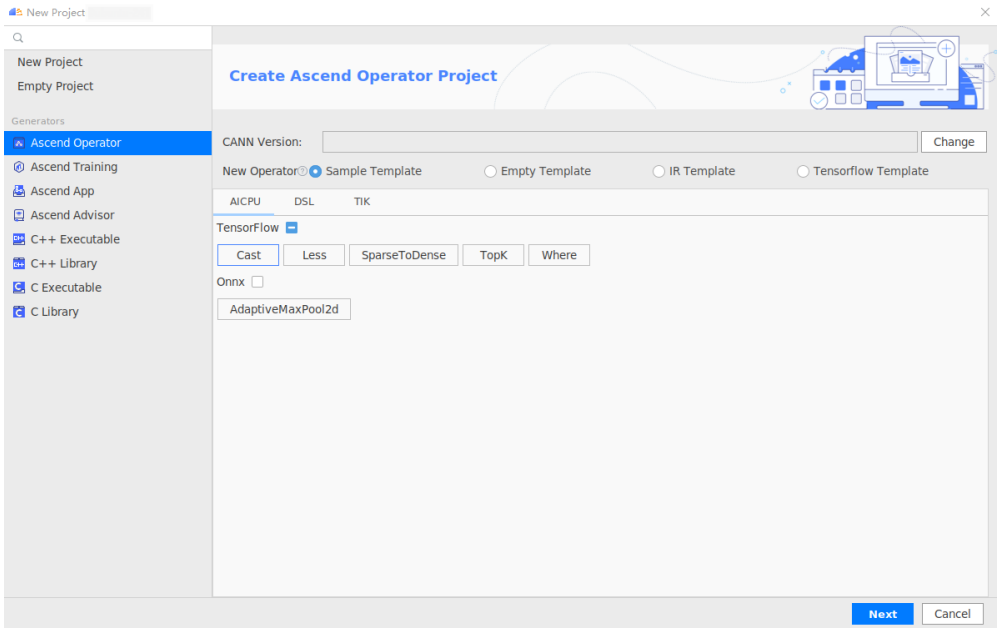
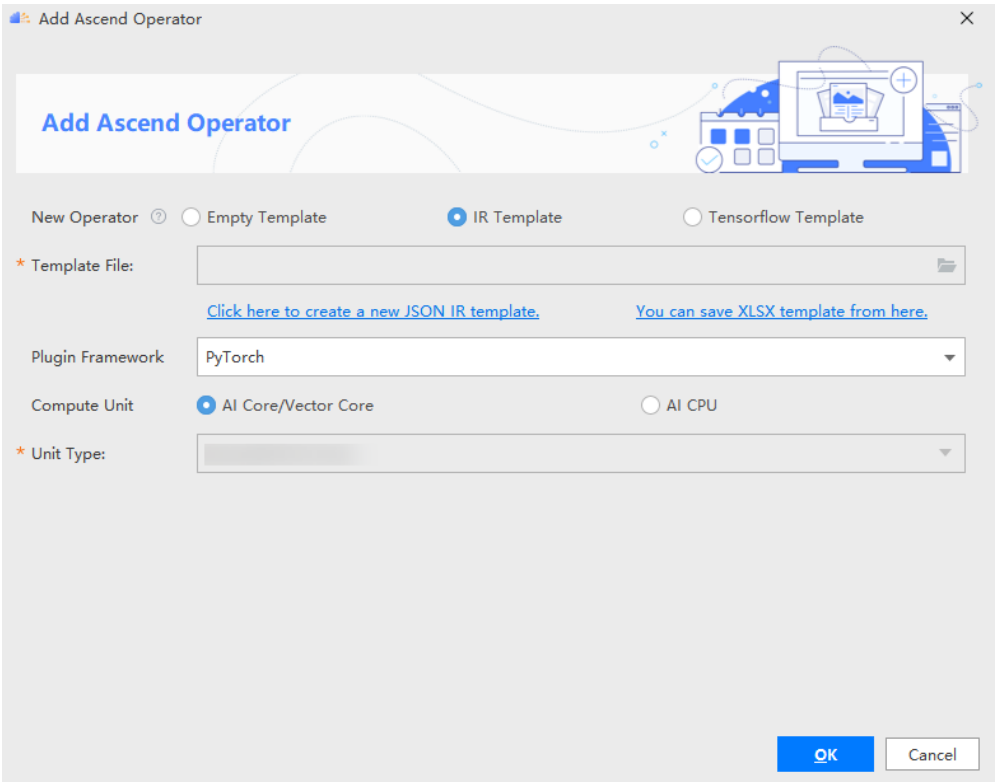


图 8-7 新建算子工程界面



步骤2 创建算子工程。

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

表 8-4 算子信息配置

参数	参数说明	示例
CANN Version	当前激活的CANN版本，可通过单击右侧“Change”进行变更，具体功能使用请参见**13.6.2 切换/激活CANN包**。	选择当前CANN的版本号
New Operator（算子创建方式）		
Sample Template	表示基于样例创建算子工程。 选择此选项，下方显示AICPU、DSL、TIK三种算子实现方式。每种实现方式按照AI框架分类提供了算子样例，供用户选择。用户可以选择一个实现方式下的一个AI框架中的一个或多个算子创建算子工程。 – AICPU：AI CPU算子样例。 – DSL：基于DSL方式实现的TBE算子样例。 – TIK：基于TIK方式实现的TBE算子样例。	四选一，当前默认选择“Sample Template”。

参数	参数说明	示例
Empty Template	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**8.4 算子分析**进行配置。	
IR Template	表示基于IR定义模板创建算子工程。IR定义模板文件有json和Excel两种格式。 选择此选项，下方会显示“Template File”配置项，用户需要选择IR原型定义文件。 **须知** – Ascend_IR_Template.json 请从*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/**python/site-packages/op_gen/json_template**目录下获取IR定义的json模板。 – Ascend_IR_Template.xlsx 请从*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/**tools/msopgen/template**目录下获取IR定义Excel模板。 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**8.13.1 IR定义配置说明**。	
Tensorflow Template	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。选择“Sample Template”创建算子工程时不显示此配置项。 – MindSpore – PyTorch – TensorFlow – Caffe – ONNX Caffe不支持Tensorflow Template创建算子工程。	MindSpore
Compute Unit	有以下两种选项，选择“Sample Template”创建算子工程时不显示此配置项。 – 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

参数	参数说明	示例
Unit Type	请根据实际昇腾AI处理器版本下拉选择算子计算单元。 – 当“Compute Unit”选择“AI Core/Vector Core”时显示此配置项。 – 当选择“Sample Template”创建算子工程时不显示此配置项。 – “Plugin Framework”选择“MindSpore”时不显示此配置项。	选择当前昇腾AI处理器的版本

2. 单击“Next”，在弹出的页面中配置算子工程信息，如表8-5所示。

表 8-5 工程信息配置

参数	参数说明	示例
Project name	工程名称，用户自行配置。 名称开头和结尾必须是数字或字母，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。	untitled
Project location	工程默认保存路径，用户可自定义。（对于首次使用MindStudio的用户，该项默认为“*$HOME*/MindstudioProjects”。）	保持默认
More Settings	“Module name”：模块名，默认与“Project name”一致。	保持默认
	“Content root”：根目录下路径。	
	“Module file location”：模块文件路径。	
	单击“Project format”右侧选框，出现下拉菜单。 – .idea（directory-based）：创建项目的时候创建一个.idea项来保存项目的信息，默认选项。 – .ipr（file-based）：项目配置文件来保存项目的配置信息。	

3. 单击“Create”，完成算子工程的创建。
   若工作窗口已打开其他工程，会出现确认提示。
   – 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。

须知

- 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和编译。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AI CPU算子和TBE算子同名。
- 参考**步骤1**在已有的MindSpore AI CPU算子工程中快捷新建算子工程时，仅支持新建MindSpore框架的AI CPU算子工程。
- 目前Windows版本中，新建算子工程的代码中可能会出现无法识别的语义被红色波浪线标出，但不影响正常开发流程，用户可忽略该提示。
- 目前Windows版本中，支持实现算子原型文件和AI CPU算子代码实现的C++代码跳转功能。需要用户手动配置系统环境变量。

  变量名为ASCEND_OPP_PATH，变量值为C:\Users*用户名*\.mindstudio\huawei\adk\remote*CANN包版本号*\opp，请以实际**用户名**和**CANN包版本号**修改变量值。
- 若使用远程进行算子开发，支持自动同步文件功能。
- 基于“Sample Template”中非MindSpore框架的样例创建算子工程时，若算子计算单元为Atlas 推理系列产品或Atlas A2 训练系列产品系列的算子计算单元，在op_info_cfg算子信息库文件目录下仅生成“ascend*xxx*”目录。
- 基于“Empty Template”、“IR Template”和“Tensorflow Template”方式创建算子工程时，将根据已选择“Unit Type”的算子计算单元在op_info_cfg算子信息库文件目录下生成对应目录。

步骤3 查看算子工程目录结构和主要文件。

框架为TensorFlow、PyTorch、ONNX、Caffe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.cc   //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
│   ├── caffe_plugin                   //存放caffe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
│   │   ├── caffe_add_plugin.cc    //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
│   │   └── onnx_plugin   //原始框架类型为ONNX时生成的算子适配插件代码所在目录
│   │       ├── CMakeLists.txt
│   │       └── xxx_plugin.cc/    /算子适配插件实现文件
│   ├── CMakeLists.txt                 //编译规则文件，会被算子工程根目录中的CMakeLists.txt文件调
├── op_proto                          //算子IR定义文件目录
│   ├── add.cc                       //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
├── op_tiling  //算子tiling实现文件目录，不涉及tiling实现的算子无需关注
│   ├── CMakeLists.txt
├── 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
│   │   ├── vector_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CANN
版本
├── Add_README.md                               //介绍算子基本功能、基本信息和主要的算子工程
├── CMakeLists.txt
├── MyOperator.iml
```

框架为MindSpore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 TBE算子工程不会创建此目录
│   ├── impl                            //算子实现文件目录
│   │   ├── cus_square_kernel.cc        //算子源文件
│   │   ├── cus_square_kernel.h         //算子头文件
│   ├── op_info_cfg                     //算子信息库文件目录
│   │   ├── aicpu_kernel
│   │   │   ├── cus_square.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
│   ├── 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CANN版本
├── cus_square_README.md        //介绍算子基本功能、基本信息和主要的算子工程
├── MyOperator.iml
```

----结束

8.6 算子开发过程

8.6.1 算子原型定义

简介

开发者进行算子原型定义开发时需要实现如下两个文件：

- 在*算子名称*.h头文件中进行算子原型的注册。
- 在*算子名称*.cc文件中进行校验函数与shape推导函数的实现。

本章节将以TBE算子开发样例工程中“Add”算子为例讲解算子原型定义的开发，供开发者参考。

进入“op_proto”目录，编写IR实现文件“add.h”和“add.cc”，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。

说明

AI CPU算子原型定义的开发可参考算子开发样例工程中“reshape_cust”算子，IR实现文件为“reshape_cust.h”和“reshape_cust.cc”。

add.h 实现

MindStudio已在add.h头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_       //条件编译
#define GE_OPS_OP_PROTO_ADD_H_       //进行宏定义
#include "graph/operator_reg.h"
namespace ge {
REG_OP(Add)     //算子类型名称
    .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，第三方框架（TensorFlow/ONNX/Caffe）需要与**8.6.5 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**8.6.2 算子代码实现（TBE DSL）**函数形参顺序以及**8.6.4 算子信息库定义**中参数的顺序保持一致。

add.cc 实现

开发者需要在add.cc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.cc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，Infershape函数注册
COMMON_INFER_FUNC_REG(Add, AddInferShape);      //第一个参数为算子的OpType


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


8.6.2 算子代码实现（TBE DSL）

简介

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 tbe.dsl as tbe
from tbe import tvm
from tbe.common.register import register_op_compute
from topi import generic


# 算子计算函数
@register_op_compute(("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。
  - “tbe.dsl”：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/latest/**compiler/python/site-packages/tbe/dsl**目录下的Python函数。
  - “tbe.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/latest/**compiler/python/site-packages/tbe/tvm**目录下的Python函数，使用方法请参见**https://docs.tvm.ai/**。

  - “tbe.common.register.register_op_compute”：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/latest/**compiler/python/site-packages/tbe/common/register/register_api.py**文件中的register_op_compute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“Sample Template”或者“Tensorflow Template”，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 Template”，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**tbe.dsl.vadd( )**要求两个输入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：

说明

若开发者通过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**14.3.2 Windows上远程打开MindStudio时，复制的内容无法粘贴到编辑器窗口中**。

```
from __future__ import absolute_import
import tbe.dsl as tbe
from functools import reduce
from tbe import tvm
from tbe.common.register import register_op_compute
from tbe.common.utils import para_check
from tbe.common.utils import shape_util


SHAPE_SIZE_LIMIT = 2147483648

# 实现Add算子的计算逻辑
@register_op_compute("Add", op_mode="dynamic", support_fusion=True)
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**8.6.4 算子信息库定义**“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**8.6.4 算子信息库定义**“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**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。

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：

```
@register_op_compute("Add", op_mode="dynamic", support_fusion=True)
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。

   ```
   @register_op_compute("Add", op_mode="dynamic", support_fusion=True)
   def add_compute(input_x, input_y, output_z, kernel_name="add")
   ```

   装饰器@register_op_compute("Add", op_mode="dynamic", support_fusion=True)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

8.6.3 算子代码实现（AI CPU）

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 _AICPU_RESHAPE_CUST_KERNELS_H_
#define _AICPU_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，包含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AI CPU的Tensor类的定义及相关方法。
  - 头文件cpu_tensor_shape.h，包含AI CPU的TensorShape类及相关方法。
  - 头文件cpu_types.h，包含AI CPU的数据类型以及格式等定义。
  - 头文件cpu_attr_value.h，包含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，包含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AI CPU的Tensor类的定义及相关方法。
   - 头文件cpu_tensor_shape.h，包含AI CPU的TensorShape类及相关方法。
   - 头文件cpu_types.h，包含AI CPU的数据类型以及格式等定义。
   - 头文件cpu_attr_value.h，包含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*RESHAPE_CUST*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 *inputTensor = 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 (inputTensor == nullptr) {
        return -1;
    }

    Tensor *outputTensor = 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 (outputTensor == nullptr) {
        return -1;
    }

    // 获取输入tensor的数据地址
    auto inputData = inputTensor->GetData();
    if (inputData == nullptr) {
        return -1;
    }

    // 获取输出tensor的数据地址
    auto outputData = outputTensor->GetData();
    if (outputData == nullptr) {
        return -1;
    }

    // 将输入tensor的数据拷贝至输出tensor中
    uint64_t inputDataSize = inputTensor->GetDataSize();
    memcpy(outputData, inputData, inputDataSize);
    return 0;
}
```


8.6.4 算子信息库定义

简介

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处理器上的具体实现规格。

针对TBE算子，算子信息库包括算子支持的输入输出dtype、format以及输入shape等信息；针对AI CPU算子，算子信息库包括算子输入输出的name，支持的dtype、format等信息。

网络运行时，图编译器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。

TBE 算子信息库

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。

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8-6**。下表仅列出常用的算子信息库配置项，其他配置项请参见《**TBE&AI CPU算子开发指南**》的“算子开发过程 > 算子信息库定义 > TBE算子信息库”章节。

表 8-6 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**8.4 算子分析**，算子类型为Add。

信息	Add算子配置	说明
input0.name	x1	Add算子的第一个输入tensor的名称，根据**8.4 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.format	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	定义输入tensor支持的数据类型与数据排布格式。 根据**8.4 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。 **说明** 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**8.4 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。

信息	Add算子配置	说明
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**8.4 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。 **说明** 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。
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**8.4 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**8.4 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。 **说明** 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。
output0.shape	all	定义输出tensor支持的形状。

信息	Add算子配置	说明
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。

AI CPU 算子信息库

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8-7**。下表仅列出常用的算子信息库配置项，其他配置项请参见《**TBE&AI CPU算子开发指南**》的“算子开发过程 > 算子信息库定义 > AI CPU算子信息库”章节。

表 8-7 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**8.4 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

8.6.5 算子适配插件实现

8.6.5.1 TensorFlow/Caffe 框架

开发者需要进行算子适配插件的开发，实现将TensorFlow/Caffe框架网络中的算子进行解析并映射成昇腾AI处理器中的算子。

本章节将以TBE算子开发样例工程中“Add”算子为例讲解算子适配插件的实现，供开发者参考。

MindStudio在“framework/tf_plugin/tensorflow_add_plugin.cc”（以TensorFlow为例）文件已自动生成了Add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安装目录/ascend-toolkit/latest/compiler/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代表原始框架类型为TensorFlow，CAFFE代表原始框架为Caffe。
  - OriginOpType：算子在原始框架中的类型。
  - 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，需要手动添加。ImplyType::TVM表示该算子是TBE算子，ImplyType::AI_CPU表示该算子是AI CPU算子。

8.6.5.2 ONNX 框架

开发者需要进行算子适配插件的开发，实现将ONNX框架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/onnx_plugin/xxx_plugin.cc”文件已自动生成了算子的插件代码。

- 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安装目录/ascend-toolkit/latest/compiler/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 {
// Onnx ParseParams
Status ParseParamAdd(const Message* op_src, ge::Operator& op_dest) {
    // To do: Implement the operator plugin by referring to the Onnx Operator Development Guide.
    return SUCCESS;
}

// register op info to GE
REGISTER_CUSTOM_OP("Add")
    .FrameworkType(ONNX)   // Operator name with the original framework
    .OriginOpType("")      // Set the original frame type of the operator
    .ParseParamsByOperatorFn(ParseParamAdd)// Registering the callback function for parsing operator parameters
    .ImplyType(ImplyType::TVM);    // 需手动添加，TBE算子：ImplyType::TVM；AI CPU算子：ImplyType::AI_CPU
} // namespace domi
```
  - REGISTER_CUSTOM_OP：注册自定义算子，*Add*作为注册到GE中的算子类型，可以任意命名但不能和已有的算子命名冲突，且需要与原型注册中的*OpType*保持一致。

- FrameworkType：ONNX代表原始框架为ONNX。
- OriginOpType：算子在原始框架中的类型，需要用户填写。例如自定义算子Add，对应ONNX算子库版本opset_version=11的原始框架类型为“ai.onnx::11::Add”，此处填写为OriginOpType("ai.onnx::11::Add")。
- ParseParamsByOperatorFn(*ParseParamAdd*)：用来注册解析算子属性的函数，需要用户自定义实现回调函数ParseParamAdd。

  回调函数ParseParamAdd的声明如下所示：

  ```
  Status ParseParamAdd(const ge::Operator& op_src, ge::Operator& op_dest)
  ```

  - *ParseParamAdd*：函数名称，用户自定义，需要保持唯一。
  - op_src：ONNX框架定义的Operator类对象，包含ONNX模型中自定义的算子属性信息，定义来源ONNX框架的原始模型文件。
  - op_dest：CANN算子数据结构，保存算子信息。
- ImplyType：指定算子的实现方式，需要手动添加。ImplyType::TVM表示该算子是TBE算子，ImplyType::AI_CPU表示该算子是AI CPU算子。

8.7 UT 测试

8.7.1 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正常运行，选取的场景组合应能覆盖算子代码的所有分支（一般情况下覆盖率要达到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/latest/**python/site-packages/op_test_frame/ut/op_ut.py**文件。

8.7.2 TBE 算子 UT 测试

前提条件

需完成自定义算子的开发，包括算子实现代码和算子原型定义，详情可参见**8.6.2 算子代码实现（TBE DSL）**和**8.6.1 算子原型定义**。

说明

- CentOS 7.8 arm容器暂不支持算子实现代码的UT测试功能。
- Windows版本的MindStudio UT测试的算子工程目录中不要出现空格。

生成 UT 测试用例文件

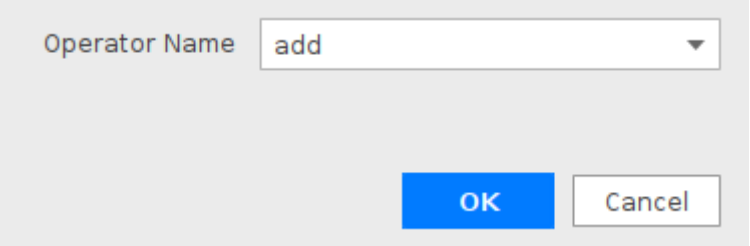
步骤1 创建UT测试用例。

1. 创建UT测试用例，有以下两种方式：

   右键单击算子工程根目录，选择“New Cases > TBE 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**“testcases > ut”**目录，选择“New Cases > TBE 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?”。

可以选择“Overwrite”覆盖当前测试用例或者“Cancel”取消覆盖。

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.cc     //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.cc             //测试用例调用总入口
│   │   │   ├ CMakeLists.txt
```

步骤2 在中标麒麟和银河麒麟运行时，请在“testcases/ops_test/add/CMakeLists.txt”文件中添加**加粗**内容，其他操作系统请忽略此步骤。

```
add_definitions(-D_GLIBCXX_USE_CXX11_ABI=0)
set(CMAKE_CXX_FLAGS "-std=c++11")
set(PROJECT_DIR "$ENV{PROJECT_PATH}")
set(GTEST_DIR ${PROJECT_DIR}/testcases/libs/gtest)
set(ADK_DIR "$ENV{ADK_PATH}")
set(ATC_DIR ${ADK_DIR}/atc)
set(OP_PROTO_SRC_DIR ${PROJECT_DIR}/op_proto)

message(STATUS "ATC_DIR=${ATC_DIR}")

enable_testing()

include_directories(
        "${GTEST_DIR}/include"
        "${ATC_DIR}/include"
        "${OP_PROTO_SRC_DIR}"
        )

aux_source_directory(${OP_PROTO_SRC_DIR} OP_PROTO_SOURCE_SRCS)
file(GLOB OP_PROTO_TEST_FILES **proto.cc)
```

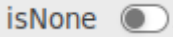


```
link_directories(
        "${ATC_DIR}/lib64"
        "${GTEST_DIR}"
        "/usr/local/gcc7.3.0/lib64/"
)

set(CUSTOM_OBJECT_NAME "add_proto_test")

add_executable(${CUSTOM_OBJECT_NAME}
        ${PROJECT_DIR}/testcases/ut/ops_test/test_main.cc ${OP_PROTO_SOURCE_SRCS} $
{OP_PROTO_TEST_FILES})

target_link_libraries(${CUSTOM_OBJECT_NAME} gtest c_sec alog pthread error_manager graph register)
```

说明

UT测试要求gcc版本为7.5.0及以上，若gcc版本不满足要求，请升级gcc版本。

步骤3 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。编写算子实现代码的UT Python测试用例的方法有以下两种：

- 根据UT_Impl模板json文件编写算子实现代码的UT Python测试用例（MindSpore算子同样支持通过UT_Impl模板编写UT Python测试用例）。
  a. 右键单击“testcases/ut/ops_test/add/test_add_impl.py”，选择“Generate TBE UT Impl Case”生成shape为空的算子UT_Impl测试用例定义文件test_add_impl.json。
  b. 在test_add_impl.json文件中，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，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**8.13.2 TBE算子UT测试用例定义文件参数解释**。若当前字段可选时，支持用户输入None字段，开启 

开关，将隐藏当前字段的配置项。
  c. UT_Impl测试用例定义文件test_add_impl.json的字段配置完成后，单击“Generate”，test_add_impl.py文件末尾会根据test_add_impl.json文件的配置生成对应的代码。
  d. 如有调用add_precision_case接口，可参考**add_precision_case接口**在test_add_impl.py文件中进行“calc_expect_func”参数配置。
- 在“testcases/ut/ops_test/add/test_add_impl.py”文件中，直接编写算子实现代码的UT Python测试用例。

```
import ...
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
})
```

说明

目前Windows版本不支持导入te或者tbe模块。故在编写算子实现代码的UT Python测试用例时不支持导入算子实现文件及涉及te或tbe模块的文件。

a. 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**8.12.4 UT测试接口参考**。

b. 实例化测试用例，OpUT的使用方法可参见**OpUT测试类定义**。

c. 用户自定义实现生成期望数据的函数。

d. 添加测试用例。

测试用例“params”中字段和字段取值范围需根据算子实现文件入口参数确定。输入tensor中的“ori_shape”和“ori_format”字段为可选字段，但若使用参数校验修饰器检验参数，“ori_shape”和“ori_format”字段必选。

可参见**8.12.4 UT测试接口参考**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4 编写算子原型定义的UT C++测试用例。

在“testcases/ut/ops_test/add/test_add_proto.cc”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);
    tensorDesc.SetOriginShape(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);
    tensorDesc1.SetOriginShape(shape1);
    tensorDesc2.SetDataType(ge::DT_FLOAT16);
    tensorDesc2.SetShape(shape2);
    tensorDesc2.SetOriginShape(shape2);

    // 更新算子输入
    operator1_op.UpdateInputDesc("x1", tensorDesc1);
    operator1_op.UpdateInputDesc("x2", tensorDesc2);
```

----结束

执行 UT 测试用例

步骤1 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。

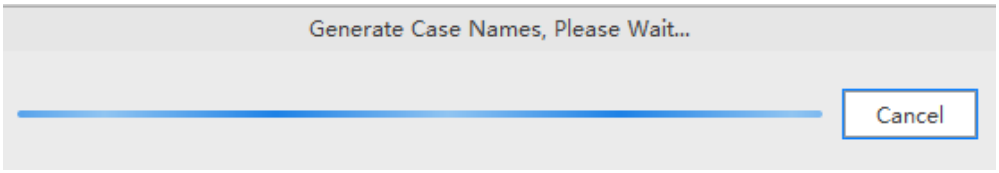
- 右键单击“testcases/ut/ops_test/*算子名称/test_算子名称_impl.py*”文件，选择“Run Tbe Operator‘算子名称’UT Impl with coverage”，执行单个算子实现代码的测试用例。

说明

Windows版本的MindStudio在进行以下操作时需获取UT Case Names，如图8-8所示将出现进度提示框（预计耗时10s左右）。

- 执行算子实现代码的测试用例时需获取UT Case Names。
- 在运行配置页面变更参数配置（Soc Version和Operator Name）时需重新获取UT Case Names。

图 8-8 获取 UT Case Names



第一次运行时会弹出运行配置页面，如图8-9所示，根据实际情况配置完成后单击“Run”。后续如需修改运行配置，请参考13.5.1 修改运行任务配置。

图 8-9 运行配置页面

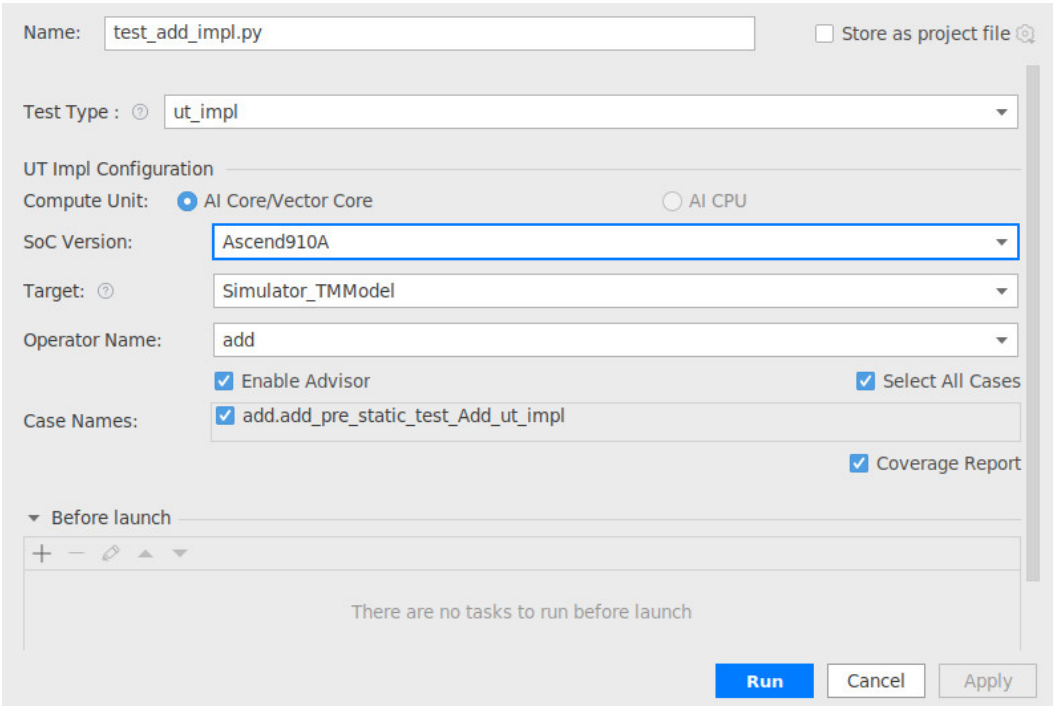


表 8-8 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_impl。

参数	说明
Compute Unit	选择计算单元，TBE算子仅支持配置AI Core/Vector Core。
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_Performance：性能仿真环境。 ● 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。**该功能仅支持Atlas 200/300/500 推理产品和Atlas 训练系列产品。**
Enable Advisor	开启专家系统。 当Target为Simulator_TMModel时，可以对单测试用例性能进行专家系统分析。专家系统请参见11.4.2 算子工程入口。
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。
CANN Machine	CANN工具所在设备的deployment信息。 **说明** 仅支持Windows操作系统。
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。支持全选或勾选部分测试用例。

步骤2 查看运行结果。

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。

1. 若配置算子期望函数"calc_expect_func": calc_expect_func，运行成功后，日志会打印对比结果。
   - Error count：误差大于绝对容忍率（atol）的数量。
   - Max atol error count：误差大于期望值 * 最大容忍率（max_atol）的数量。
   - Threshold count (rtol * data_size)：允许超过自定义精度的最大错误 (相对容忍率（rtol） * tensor元素数量)数量。

   测试结果：
   - Max atol error count>0则UT测试失败。
   - Error count>Threshold count则UT测试失败。
2. 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。

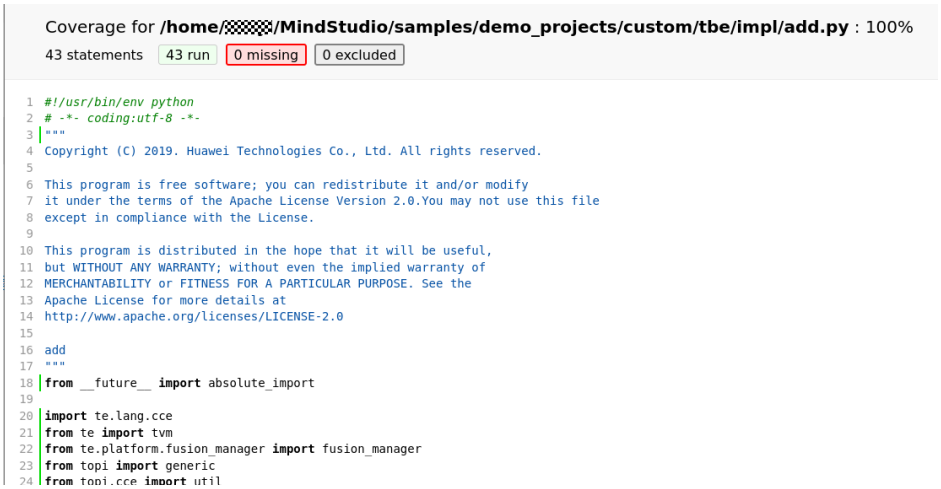
   说明

   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。

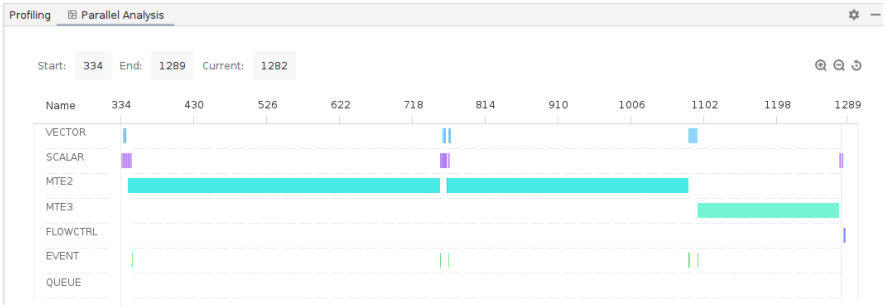
如果出现"Page'http://***.html'requested without authorization, you can copy URL and open it in browser to trust it."提示，请参考**14.3.4 配置不受信任的网址访问浏览器**解决。

3. 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8-10**所示，通过绿色和红色标签区分是否覆盖。

图 8-10 UT 覆盖率详情页面



4. 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_Performance”，可以在Profiling窗口查看执行流水线，如下图所示。

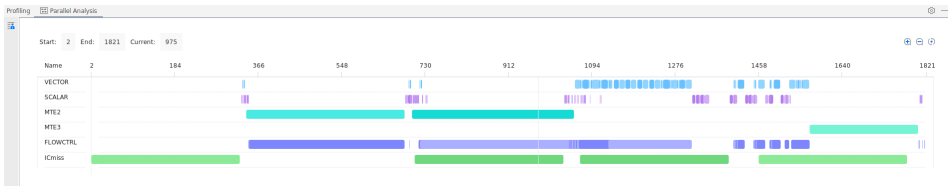


Parallel Analysis：并行度视图。并行度视图展示单算子运行中各个指令执行的并行度情况，横轴为时间（Tick，时钟周期），纵轴为各指令单元，鼠标移动到时间块上，可以显示单条指令的消耗时间以及相关指令。

说明

- “Start”和“End”表示数据展示的时间范围。“End”为Cycle数，此数值越大，代表运行时间越长。图中数据展示的时间范围可通过滚动鼠标放大或缩小进行变更。
- 右键单击时间块，弹出跳转到Profiling指令流视图和TBE、CCE、Assembly源代码菜单。

   - 如果在配置运行信息选择的“Target”为“Simulator_TMModel”，可以查看执行流水线，如下图所示。



若用户使用TIK方式实现算子开发，MindStudio UT测试支持流线图到TIK代码和CCE代码的跳转，帮助用户快速定位到流水图中对应的代码位置，配置详情请参见**设置代码跳转功能**。

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。

- 如果在配置运行信息勾选了“Enable Advisor”，可以查看专家建议，详情请参见**11.4.2.3 分析结果展示**。可以通过单击  按钮，显示和隐藏专家建议。

步骤3 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。
- 右键单击“testcases/ut/ops_test/*算子名称/test_算子名称_proto.cc*”文件，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**13.5.1 修改运行任务配置**。

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。
Test Type	选择ut_proto。
Compute Unit	选择计算单元，TBE算子仅支持配置AI Core/Vector Core。
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。支持全选或勾选部分测试用例。

步骤4 查看运行结果。

运行完成后，通过界面下方的日志打印窗口，查看运行结果。结果中展示此次运行测试用例、成功用例和失败用例的数量，如下图所示。

```
================================================================================
run command: None
--------------------------------------------------------------------------------
- test soc: [Ascend910A]
- test case count: 1
- success count: 1
- failed count: 0
- error count: 0
--------------------------------------------------------------------------------
Soc Version: Ascend910A
    success: [add]  add_pre_static_test_add_auto_case_name_1 (Ascend910A) (precision)
--------------------------------------------------------------------------------
================================================================================

end run ops ut time: 2023-06-29 11:00:52.499607
```

----结束

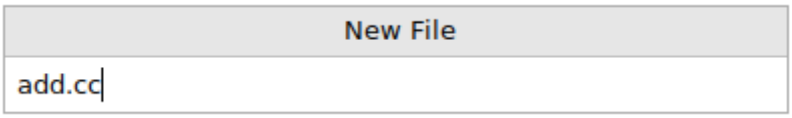
OP Tiling UT 测试

须知

动态shape算子工程涉及OP Tiling时执行以下操作，不涉及tiling实现的算子无需关注。

步骤1 创建OP Tiling的实现文件。

1. 右键单击op_tiling文件夹，选择“New > File”。
2. 在弹出的New File界面，输入*{op_name}*.cc，也可以自定义名称，需为.cc文件，如下图所示。



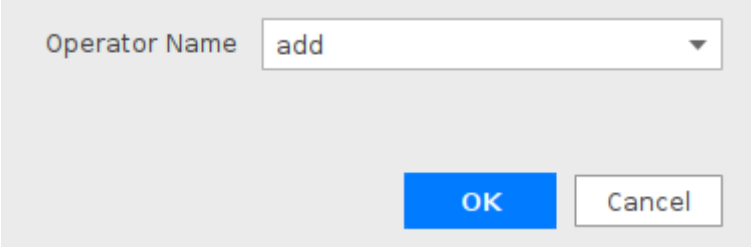
3. 回车后生成空白的*{op_name}*.cc文件，用户可根据业务需求编写该文件。

步骤2 创建OP Tiling 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 > TBE OP TILING 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 > TBE OP TILING UT Case”，创建OP Tiling UT测试用例。
2. 在弹出的算子选择界面，选择需要生成OP Tiling UT模板工程的算子，单击“OK”，如下图所示。



说明

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

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_tiling_test
│   │   │   │   ├──add
│   │   │   │   │   ├── CMakeLists.txt       //用于编译可执行文件
│   │   │   │   │   ├── test_add_tiling.cc     //算子tiling的测试用例文件
│   │   │   │   ├── CMakeLists.txt           //用于编译可执行文件
│   │   │   │   ├── test_main.cc            //测试用例调用总入口
```

步骤3 运行OP Tiling UT测试用例。

开发人员可以执行当前工程中所有算子的OP Tiling UT测试用例，也可以执行单个算子的OP Tiling UT测试用例。

- 右键单击“testcases/ut/ops_tiling_test”文件夹，选择“Run TBE Operator ‘all’ UT Tiling”，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_tiling_test/*算子名称*”文件夹，选择“Run TBE Operator‘算子名称’UT Tiling”，执行单个算子原型定义代码的测试用例。
- 右键单击“testcases/ut/ops_tiling_test/*算子名称/test_算子名称_tiling_.cc*”文件，选择“Run TBE Operator‘算子名称’UT Tiling”，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13.5.1 修改运行任务配置。

表 8-10 运行配置信息

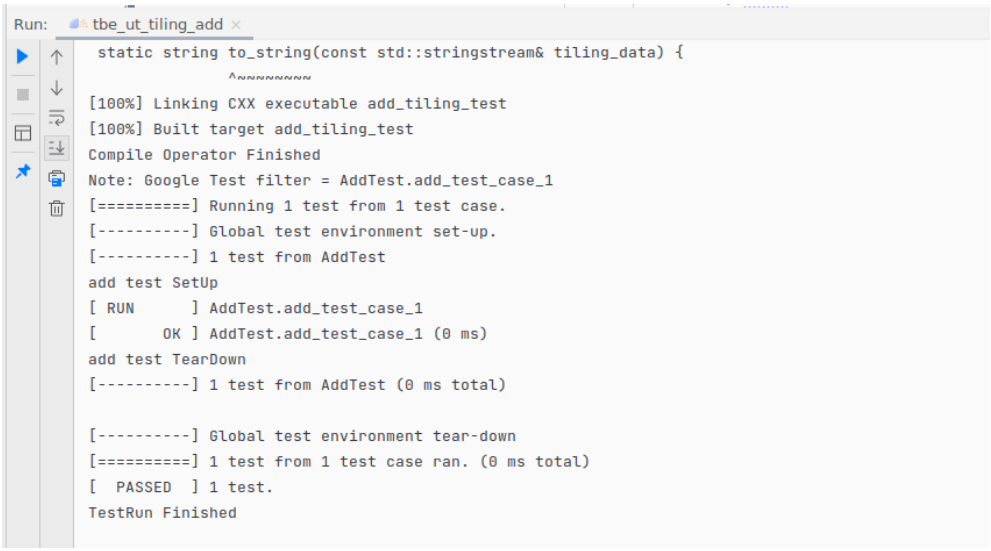
参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_tiling。
Compute Unit	计算单元，选择AI Core/Vector Core。
Operator Name	选择运行的测试用例。 • all表示运行所有用例。 • 其他表示运行某个算子下的测试用例。

参数	说明
Case Names	勾选需要运行的测试用例，即TEST_F中定义的用例。 支持全选或勾选部分测试用例。

步骤4 查看运行结果。

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。如图8-11所示。

图 8-11 运行结果



----结束

8.7.3 AI CPU 算子 UT 测试

前提条件

- 需完成自定义算子的开发，包括算子实现代码和算子原型定义，详情可参见**8.6.3 算子代码实现（AI CPU）**和**8.6.1 算子原型定义**。
- 安装lcov依赖，gcc 8及以下的版本可以安装1.14版本的lcov依赖 ，gcc 9及以上的版本建议安装1.16版本的lcov依赖。
- 安装CMake，要求版本为3.14及以上，若CMake版本不满足要求，请升级CMake版本。

说明

CentOS 7.8 arm容器暂不支持UT测试功能。

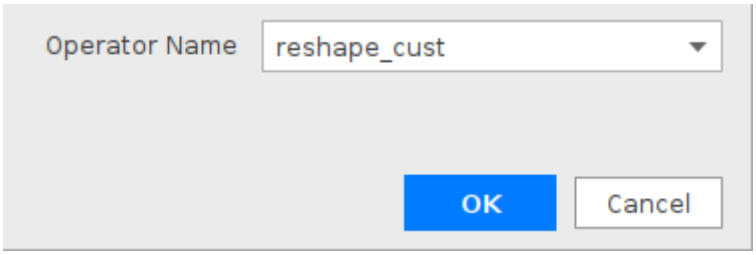
操作步骤

步骤1 创建UT测试用例。

1. 创建UT测试用例，有以下两种方式：
   右键单击算子工程根目录，选择“New Cases > AI CPU 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“**testcases > ut**”目录，选择“New Cases > AI CPU 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?”

可以选择“Overwrite”覆盖当前测试用例或者“Cancel”取消覆盖。

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
│   │   │   ├── aicpu_test
│   │   │   │   ├── reshape_cust
│   │   │   │   │   ├── CMakeLists.txt       //用于编译可执行文件
│   │   │   │   │   ├── test_reshape_cust_impl.cc     //算子实现代码的测试用例文件
│   │   │   │   │   ├── test_reshape_cust_proto.cc    //算子原型定义代码的测试用例文件，MindSpore框架的算子不包含该文件
│   │   │   │   ├── CMakeLists.txt           //用于编译可执行文件
│   │   │   │   ├── test_main.cc            //测试用例调用总入口
│   │   │   ├ CMakeLists.txt
```

步骤2 在中标麒麟和银河麒麟运行时，请在./testcases/aicpu_test/reshape_cust/CMakeLists.txt文件中添加**加粗**内容，其他操作系统请忽略此步骤。

```
cmake_minimum_required(VERSION 3.14)
set(CMAKE_CXX_FLAGS "-std=c++11")

...

link_directories(
        "${ATC_DIR}/lib64"
        "${GTEST_DIR}"
        "/usr/local/gcc7.3.0/lib64/"   //若用户自己配置dockerfile请修改为实际gcc7.3.0的lib64路径
    )

    set(CUSTOM_OBJECT_NAME "reshape_cust_proto_test")
    add_executable(${CUSTOM_OBJECT_NAME}
        ${PROJECT_DIR}/testcases/ut/aicpu_test/test_main.cc
        ${OP_PROTO_SOURCE_SRCS}
        ${OP_PROTO_TEST_FILES})

    target_link_libraries(${CUSTOM_OBJECT_NAME} gtest c_sec alog pthread error_manager graph register)
endif()
```

说明

使用AI CPU UT测试功能时，gcc版本为7.3.0及以上，若gcc版本不满足要求，请升级gcc版本。

步骤3 编写算子实现代码的UT C++测试用例。

在“testcases/ut/aicpu_test/reshape_cust/test_reshape_cust_impl.cc”文件中，编写算子实现代码的UT C++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。

步骤4 编写算子原型定义的UT C++测试用例，其中MindSpore框架不支持编写。

在“testcases/ut/aicpu_test/reshape_cust/test_reshape_cust_proto.cc”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。

步骤5 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/aicpu_test”文件夹，选择“Run AI CPU Operator ‘All’ UT Impl with coverage”，运行整个文件夹下算子实现代码的测试用例。
- 右键单击“testcases/ut/aicpu_test/*算子名称*”文件夹，选择“Run AI CPU Operator 算子名称’UT Impl with coverage” ‘，运行单个算子实现代码的测试用例。
- 右键单击“testcases/ut/ops_test/*算子名称/test_算子名称_impl.cc*”文件，选择“Run AI CPU 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**13.5.1 修改运行任务配置**。

图 8-12 运行配置页面

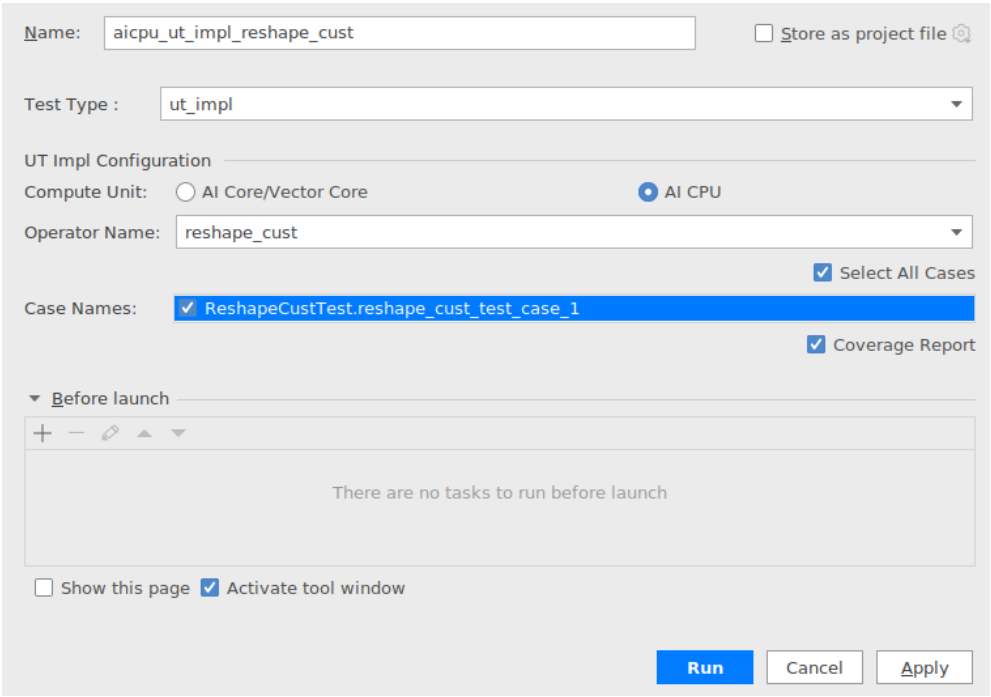


表 8-11 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_impl。
Compute Unit	选择计算单元。 • AI Core/Vector Core • AI CPU 选择不同的计算单元可以实现AI Core/Vector Core和AI CPU UT测试配置界面的切换，暂时只支持配置AI CPU。
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 C++测试用例。支持全选或勾选部分测试用例。

查看运行结果。

1. 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。如图8-13所示。

   图 8-13 运行结果

   ```
   Processing file bits/basic_string.h
   Processing file bits/move.h
   Processing file bits/basic_string.tcc
   Processing file bits/shared_ptr.h
   Processing file bits/std_function.h
   Processing file bits/char_traits.h
   Processing file bits/invoke.h
   Processing file bits/shared_ptr_base.h
   Processing file ext/atomicity.h
   Writing directory view page.
   Overall coverage rate:
     lines......: 16.0% (17 of 106 lines)
     functions..: 17.6% (3 of 17 functions)

   AICPU TestRun Finished
   ```

2. 在“Run”窗口中单击report 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。

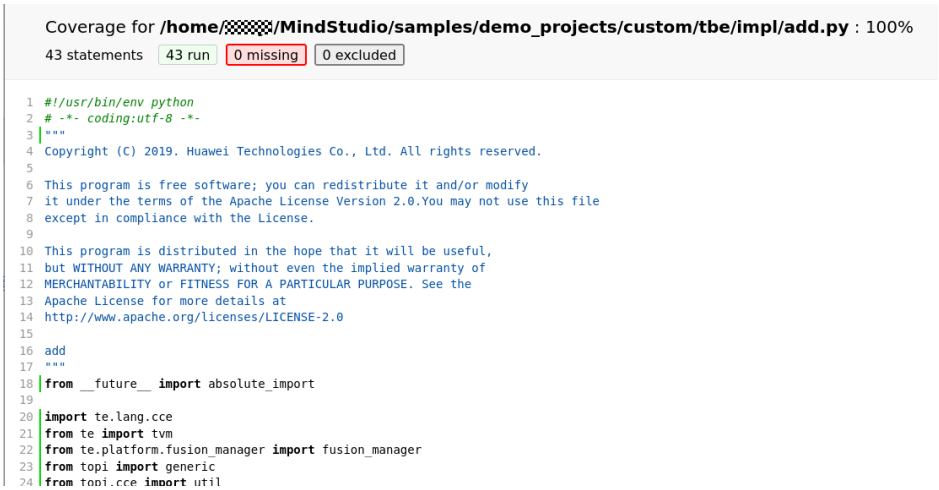
   说明

   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。

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**14.3.4 配置不受信任的网址访问浏览器**解决。

3. 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8-14**所示，通过绿色和红色标签区分是否覆盖。

图 8-14 UT 覆盖率详情页面



步骤6 （可选）调试算子实现文件的UT测试。

1. 右键单击“testcases/ut/aicpu_test/*算子名称*”文件夹，选择“Debug AI CPU Operator‘算子名称’UT Impl ”，调试单个算子实现代码的测试用例。

   调试详细操作请参见**2.3 工程调试与执行**，调试快捷键设置方法请参见**其他**。

   第一次运行时会弹出运行配置页面，请参考配置，然后单击“Debug”。后续如需修改运行配置，请参考**13.5.1 修改运行任务配置**。

表 8-12 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。
Compute Unit	计算单元，选择AI CPU。
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 C++测试用例。支持全选或勾选部分测试用例。

2. 查看调试信息。

   调试过程中，通过界面下方的“Debug”窗口进行调试、查看调试信息和调试结果。

步骤7 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/aicpu_test”文件夹，选择“Run AI CPU 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/aicpu_test/*算子名称*”文件夹，选择“Run AI CPU Operator‘算子名称’UT Proto”，执行单个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/test_算子名称_proto.cc*”文件，选择“Run AI CPU 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13.5.1 修改运行任务配置。

图 8-15 配置页面

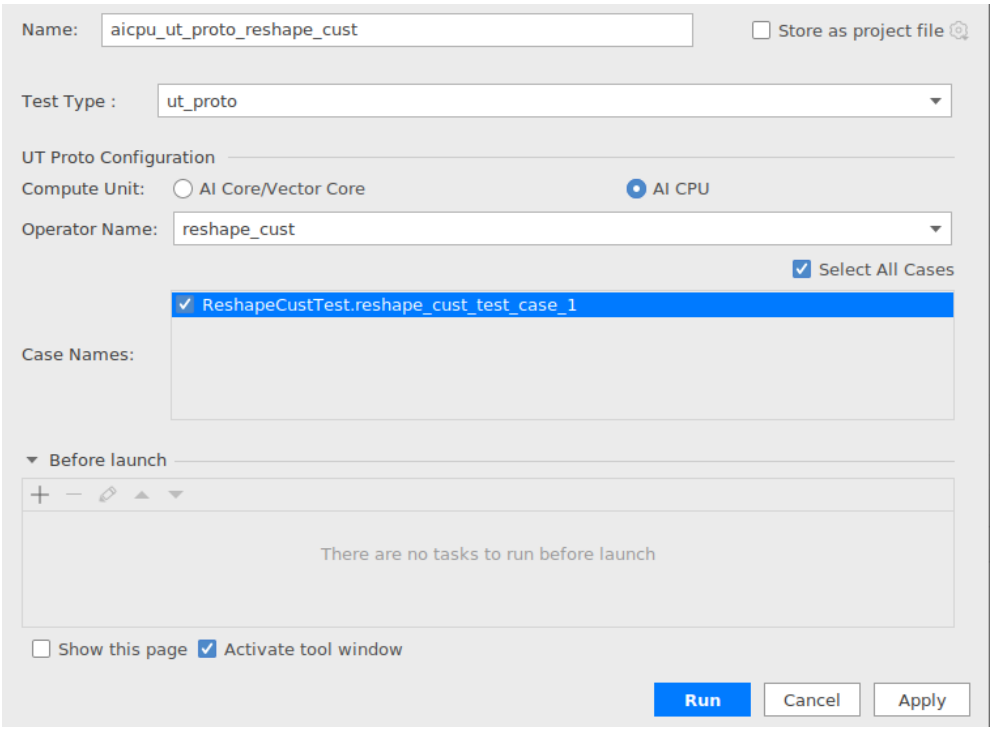


表 8-13 运行配置信息

参数	说明
Name	运行配置名称，用户可以自定义。
Test Type	选择ut_proto。

参数	说明
Compute Unit	选择计算单元。 • AI Core/Vector Core • AI CPU 选择不同的计算单元可以实现AI Core/Vector Core和AI CPU UT测试配置界面的切换，暂时只支持配置AI CPU。
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。支持全选或勾选部分测试用例。

运行完成后，通过界面下方的日志打印窗口，查看运行结果。结果中展示测试用例运行成功和失败的情况。如图8-16所示。

图 8-16 运行结果

```
Note: Google Test filter = ReshapeCustTest.reshape_cust_test_case_1
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from ReshapeCustTest
reshape_cust test SetUp
[ RUN      ] ReshapeCustTest.reshape_cust_test_case_1
[       OK ] ReshapeCustTest.reshape_cust_test_case_1 (0 ms)
reshape_cust test TearDown
[----------] 1 test from ReshapeCustTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
TestRun Finished
```

----结束

说明

旧的UT测试工程在更新Ascend-cann-toolkit开发套件包后，可能会出现UT测试运行失败。解决方法为删除旧的UT测试依赖文件夹后重新运行UT测试。旧的UT测试依赖文件夹路径为“AI CPU算子工程目录/testcases/libs/aicpu_depend”。

其他

AI CPU算子的UT测试支持单步调试，在顶部菜单栏中选择“File > Settings > Keymap > Main menu > Run > Debugging Actions”修改调试快捷键。

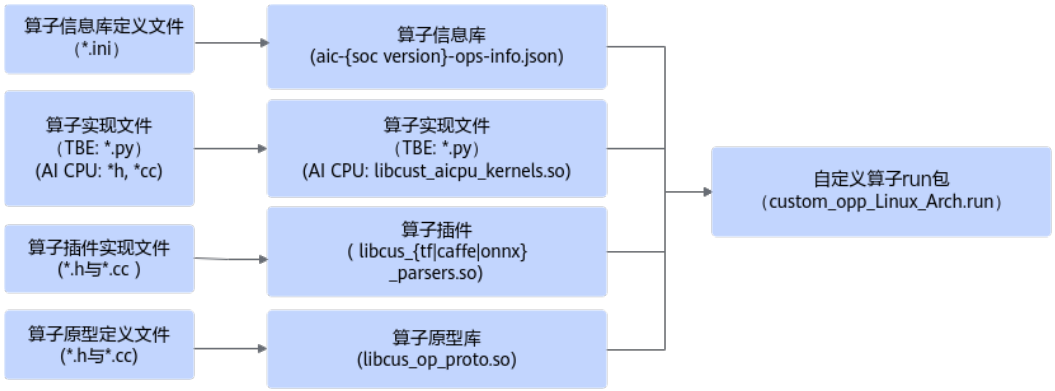
8.8 算子工程编译

简介

算子交付件开发完成后，需要对算子工程进行编译，生成自定义算子安装包**custom_opp_Linux_*Arch*.run**，详细的编译操作包括：

- 将算子信息库定义文件*.ini编译成算子信息库*.json。
- 针对AI CPU算子，将算子实现文件*.h与*.cc编译为动态库文件libcust_aicpu_kernels.so。
- 将算子插件实现文件*.h与*.cc编译成算子插件libcust_*{tf|caffe|onnx}*_parsers.so。
- 将算子原型定义文件.h与*.cc编译成算子原型库libcust_op_proto.so。

图 8-17 编译过程



说明

- Windows操作系统不支持本地编译。
- 旧版本的AI CPU算子工程在使用最新版本的Ascend-cann-toolkit开发套件包时，存在不兼容问题，需要修改CMakeLists.txt文件后重新编译算子工程。解决方法如下：
  1. 修改算子工程目录下/cpukernel/CMakeLists.txt文件。

```
...
set(AICPU_SOC_VERSION $ENV{AICPU_SOC_VERSION})
message(STATUS "AICPU_SOC_VERSION=${AICPU_SOC_VERSION}")

# 添加如下代码
if(EXISTS "${ASCEND_AICPU_PATH}/opp/built-in/op_impl/aicpu/aicpu_kernel/lib/${AICPU_SOC_VERSION}/libascend_protobuf.a")
    target_link_options(${AICPU_KERNEL_TARGET} PRIVATE
        -s
        -Wl,-Bsymbolic
        -Wl,--exclude-libs=libascend_protobuf.a
    )
    target_link_libraries(${AICPU_KERNEL_TARGET} PRIVATE
        -Wl,--whole-archive
        ${ASCEND_AICPU_PATH}/opp/built-in/op_impl/aicpu/aicpu_kernel/lib/${AICPU_SOC_VERSION}/libascend_protobuf.a
        -Wl,--no-whole-archive
    )
endif()

if(EXISTS "${ASCEND_AICPU_PATH}/opp/built-in/op_impl/aicpu/aicpu_kernel/lib/${AICPU_SOC_VERSION}/libcpu_kernels_context.a")
...
```

  2. 参考本章节重新编译算子工程。

编译操作

步骤1 （可选）AI CPU算子引用的三方库路径可通过修改算子工程目录下的“cpukernel/CMakeLists.txt”文件进行配置。

- 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
)
```

- link_libraries：添加算子实现文件的库文件。

  示例如下：

```
link_libraries(
libName1
libName2)
```

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，选中算子工程，单击顶部菜单栏的“Build > Edit Build Configuration...”。

步骤3 进入编译配置页面。单击 + 添加新增配置，默认添加编译类型“Release(default)”，请参考**表8-14**进行编译配置。

图 8-18 编译配置页面

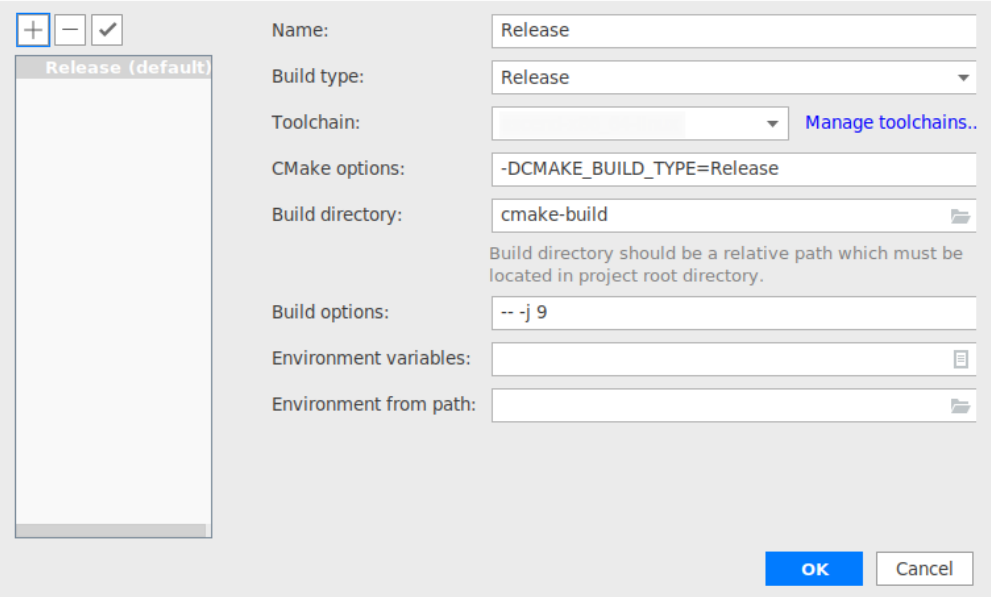


表 8-14 编译配置参数说明

参数	说明
Name	配置名称自定义，默认为Release。
Build type	配置类型，可选，默认为Release。
Toolchain	工具链配置器，根据已安装的CANN包预置架构一致的自定义Toolchain，支持本地和远程编译功能。 可单击“Manage toolchains...”自定义配置Toolchain，配置详情请参见**13.4 Toolchains**。
CMake options	CMake选项，默认：“-DCMAKE_BUILD_TYPE=Release”。
Build directory	编译目录相对路径，该路径是相对于工程目录的路径。
Build options	编译加速选项。
Environment variables	环境变量配置：支持编译前配置环境变量。 可直接手动配置或单击 符号，在弹出窗中配置管理。
Environment from path	输入路径或单击右侧，选择环境变量配置文件。配置文件以单行*<变量名>=<变量值>*方式填写，保存为文件，如： APATH=/usr/local/xxx X_PATH=/xxx/xxx

若选用远程Toolchain，将默认添加一个Deployment，用户需要配置环境变量。

在Environment Variables输入框中输入环境变量。

- ASCEND_TENSOR_COMPILER_INCLUDE：CANN软件头文件所在路径。
  ASCEND_TENSOR_COMPILER_INCLUDE=*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/include
- ASCEND_OPP_PATH：用于查找AI CPU算子头文件所在的路径编。
  ASCEND_OPP_PATH=*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/opp;
- ASCEND_AICPU_PATH：用于查找AI CPU算子相关的静态库。
  ASCEND_AICPU_PATH=*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest
- AICPU_KERNEL_TARGET：AI CPU算子实现文件编译生成的动态库文件名称。
  AICPU_KERNEL_TARGET=cust_aicpu_kernels_*3.3.0*

  说明

  建议用户配置AICPU_KERNEL_TARGET环境变量，并添加软件版本号的后缀，避免后续由于AI CPU软件升级造成自定义AI CPU动态库文件的冲突。

  若不配置此环境变量，使用默认值：cust_aicpu_kernels。

  若选用本地编译需用户在操作系统中增加*环境变量*。
  export ASCEND_AICPU_PATH=*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest
  export AICPU_KERNEL_TARGET=cust_aicpu_kernels_*3.3.0*（建议配置）

步骤4 单击 或“Build > Build Ascend Operator Project”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_Linux_*Arch*.run**。

其中**Arch**的取值根据安装的CANN包和Toolchain的信息获取。

----结束

8.9 算子部署

8.9.1 本地部署

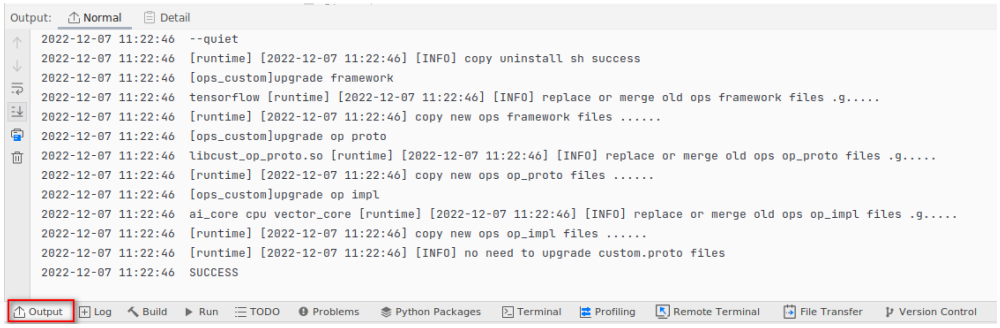
须知

- 执行**8.11 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**8.11 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**8.8 算子工程编译**生成自定义算子安装包**custom_opp_Linux_*Arch*.run**。
- Windows操作系统不支持算子本地部署。

步骤1 在MindStudio工程界面菜单栏依次选择“Ascend > Operator Deployment”，弹出算子部署界面。

步骤2 在弹出的界面中选择“Operator Deploy Locally”，在“Operator Package”中选择指定的算子库OPP包目录并单击“Operator deploy”。

在下方Output页签出现如下信息，代表自定义算子部署成功。



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/latest/**opp**目录下或指定的算子库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
│   ├── vendors    //自定义算子所在目录
│   │   ├── config.ini     // 自定义算子优先级配置文件
│   │   ├── vendor_name1    // 存储对应厂商部署的自定义算子，此名字为编译自定义算子安装包时配置的vendor_name，若未配置，默认值为customize
│   │   │   ├── op_impl
│   │   │   │   ├── ai_core    //TBE算子实现文件及算子信息库所在目录
│   │   │   │   │   ├── tbe
│   │   │   │   │   │   ├── config
│   │   │   │   │   │   │   ├── soc_version     //昇腾AI处理器版本
│   │   │   │   │   │   │   │   ├── aic-soc_version-ops-info.json     //TBE算子信息库文件
│   │   │   │   │   │   ├── impl              //TBE算子实现代码文件
│   │   │   │   │   │   │   ├── add.py
│   │   │   │   ├── cpu    //AI CPU算子实现文件及算子信息库所在目录
│   │   │   │   │   ├── aicpu_kernel/
│   │   │   │   │   │   ├── impl              //AI CPU算子实现代码文件
│   │   │   │   │   │   │   ├── libcust_aicpu_kernels.so
│   │   │   │   │   ├── config
│   │   │   │   │   │   ├── cust_aicpu_kernel.json     //AICPU算子信息库文件
│   │   │   │   ├── vector_core    //此目录预留，无需关注
│   │   │   ├── framework
│   │   │   │   ├── caffe    //存放Caffe框架的自定义算子插件库
│   │   │   │   │   ├── libcust_caffe_parsers.so
│   │   │   │   │   ├── custom.proto
│   │   │   │   ├── onnx          //存放ONNX框架的自定义算子插件库
│   │   │   │   │   ├── libcust_onnx_parsers.so
│   │   │   │   ├── tensorflow          //存放TensorFlow框架的自定义算子插件库及npu对相关自定义算子支持度的配置文件
│   │   │   │   │   ├── libcust_tf_parsers.so
│   │   │   │   │   ├── libcust_tf_scope_fusion.so    //scope融合规则定义库文件
│   │   │   │   │   ├── npu_supported_ops.json    //Atlas 训练系列产品使用的文件
│   │   │   ├── op_proto    //自定义算子原型库所在目录
│   │   │   │   ├── libcust_op_proto.so
│   │   ├── vendor_name2    // 存储厂商vendor_name2部署的自定义算子
```

----结束

8.9.2 远程部署

将自定义算子安装包**custom_opp_Linux_*Arch*.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 > Operator Deployment”，进入算子打包部署界面。

请在“Operator Deploy Remotely > Deployment”选择配置项，配置“Deployment”的方法请参见**13.3 Ascend Deployment**。

步骤3 配置环境变量，两种配置方式如下所示：

- 在昇腾AI处理器所在硬件环境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=Ascend-cann-toolkit安装目录/ascend-toolkit/latest/opp
  ```
  Ascend-cann-toolkit安装目录/ascend-toolkit/latest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=*Ascend-cann-toolkit安装目录/ascend-toolkit/latest/opp*。

  Ascend-cann-toolkit安装目录/ascend-toolkit/latest为OPP组件（算子库）的安装路径，请根据实际情况配置。

  也可以单击文本框后的图标，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：*Ascend-cann-toolkit安装目录/ascend-toolkit/latest/opp*。

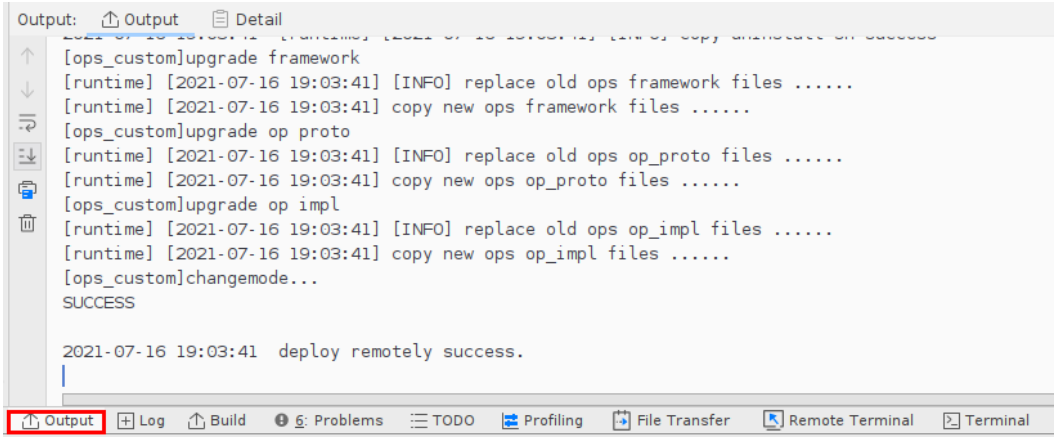
步骤4 选择指定的算子库OPP包。

在“Operator Package”中选择指定的算子库OPP包目录。

步骤5 选择算子部署的目标服务器，单击“Operator deploy”。

步骤6 算子部署过程即**8.8 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，若不选择指定的算子库OPP包则默认路径为/usr/local/Ascend/opp/。

图 8-19 算子部署日志打印



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
│   ├── vendors    //自定义算子所在目录
│   │   ├── config.ini     // 自定义算子优先级配置文件
│   │   ├── vendor_name1   // 存储对应厂商部署的自定义算子，此名字为编译自定义算子安装包时配置的
```



```
vendor_name，若未配置，默认值为customize
    ├── op_impl
    │   ├── ai_core
    │   │   ├── tbe
    │   │   │   ├── config
    │   │   │   │   ├── aic_ops_info.json      //自定义算子信息库文件
    │   │   │   ├── vendor_name1_impl          //自定义算子实现代码文件
    │   │   │   │   ├── add.py
    │   │   ├── cpu
    │   │   │   ├── aicpu_kernel/
    │   │   │   │   ├── vendor_name1_impl              //自定义算子实现代码文件
    │   │   │   │   │   ├── libcust_aicpu_kernels.so
    │   │   │   ├── config
    │   │   │   │   ├── cust_aicpu_kernel.json     //自定义算子信息库文件
    │   │   ├── vector_core     //此目录预留，无需关注
    │   ├── framework
    │   │   ├── caffe        //存放Caffe框架的自定义算子插件库
    │   │   ├── onnx         //存放ONNX框架的自定义算子插件库
    │   │   ├── tensorflow          //存放TensorFlow框架的自定义算子插件库
    │   │   │   ├── libcust_tf_parsers.so
    │   │   │   ├── npu_supported_ops.json   //Atlas 训练系列产品使用的文件
    │   ├── op_proto
    │   │   ├── libcust_op_proto.so    //自定义算子原型库文件
```

----结束

8.10 PyTorch 适配

说明

MindStudio暂不支持PyTorch算子适配，在完成TBE或AI CPU算子开发后，PyTorch适配请参见《**PyTorch算子开发指南**》。

PyTorch适配请读者根据《**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或AI CPU算子

8.11 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：

- 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。
- 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设计算子的测试用例。
- 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

- 完成自定义算子如下交付件的开发：**8.6.2 算子代码实现（TBE DSL）**/**8.6.3 算子代码实现（AI CPU）**、**8.6.1 算子原型定义**和**8.6.4 算子信息库定义**，ST测试不会对算子适配插件进行测试。
- 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，有以下三种方式：

- 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创建ST测试用例。

  TBE算子：***{工程名}*** **/tbe /op_info_cfg/ai_core/*{SoC version}*** **/*xx*.ini**，选择“New Cases > ST Case”。

  AI CPU算子：***{工程名}*** **/cpukernel/op_info_cfg/aicpu_kernel/*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 Name：下拉选择算子名称。
- SoC Version：下拉选择昇腾AI处理器的类型。若对AI CPU算子进行ST测试，默认选择aicpu_kernel。
- 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，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，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**8.13.3 TBE算子ST测试用例定义文件参数解释**或**8.13.4 AI CPU算子ST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）或者onnx格式的模型文件后，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。

  此文件用于生成测试数据及测试用例，可以参见**8.13.3 TBE算子ST测试用例定义文件参数解释**或**8.13.4 AI CPU算子ST测试用例定义文件参数解释**进行相关字段的修改。

说明

该功能需要在运行环境中安装第三方框架（PyTorch使用该功能需要在运行环境中安装onnx库）。若为Windows操作系统则需要在Windsows本地安装TensorFlow框架和ONNX库。

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。

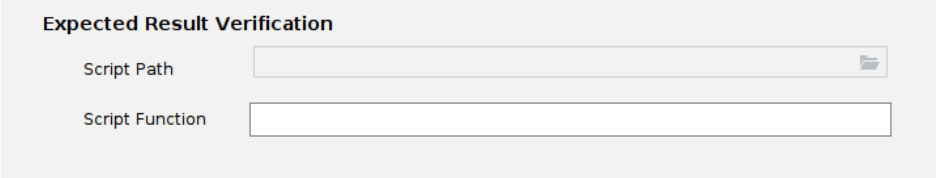
1. 自定义实现算子期望数据生成函数。

   算子期望数据生成函数需要开发者使用Python语言自行实现，用于在CPU上运行生成标杆数据。例如，开发者可以使用numpy等接口实现与自定义算子功能相同的函数。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”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。

     示例：test_add.py文件为add算子期望数据生成文件，函数实现如下：

     ```
     def calc_expect_func(input_x, input_y, out):
         res = input_x["value"] + input_y["value"]
         return [res, ]
     ```

     说明

     用户需根据开发的自定义算子完成算子期望数据生成函数。测试用例定义文件中的全部Input、Output、Attr的name作为算子期望数据生成函数的输入参数。

步骤4 若要使用python脚本批量生成测试用例，采用fuzz方式生成测试用例。

1. 实现fuzz测试参数生成脚本。该脚本可以自动生成测试用例定义文件中Input[xx]、Output[xx]、Attr内除了name的任何参数。

下面以随机生成shape和value参数为例，创建一个fuzz_shape.py供用户参考。该示例会随机生成一个1-4维，每个维度取值范围在1-64的shape参数，用于ST测试。

a. 导入脚本所需依赖。

```
import numpy as np
```

b. 实现fuzz_branch()方法，若用户自定义随机生成待测试参数的方法名，需要在算子测试用例定义文件中**Fuzz Function**字段配置自定义方法。

```
def fuzz_branch():
    # 生成测试参数shape值
    dim = random.randint(1, 4)
    x_shape_0 = random.randint(1, 64)
    x_shape_1 = random.randint(1, 64)
    x_shape_2 = random.randint(1, 64)
    x_shape_3 = random.randint(1, 64)
    if dim == 1:
        shape = [x_shape_0]
    if dim == 2:
        shape = [x_shape_0, x_shape_1]
    if dim == 3:
        shape = [x_shape_0, x_shape_1, x_shape_2]
    if dim == 4:
        shape = [x_shape_0, x_shape_1, x_shape_2, x_shape_3]
    # 根据shape随机生成x1、x2的value
    fuzz_value_x1 = np.random.randint(1, 10, size=shape)
    fuzz_value_x2 = np.random.randint(1, 10, size=shape)

    # 用字典数据结构返回shape值，将生成的shape值返回给input_desc的x1、x2和output_desc的y的shape参数。其中x1、x2、y测试用例定义文件输入、输出的name。
    return {"input_desc": {"x1": {"shape": shape,"value": fuzz_value_x1},
                           "x2": {"shape": shape,"value": fuzz_value_x2}},
            "output_desc": {"y": {"shape": shape}}}
```

- 该方法生成测试用例定义文件Input[xx]、Output[xx]、Attr内除了name的任何参数，用户可自定义实现参数的生成方法，以满足算子测试的需求。
- 该方法的返回值为字典格式，将该方法生成的参数值以字典的方式赋值给算子进行st测试。返回的字典结构与测试用例定义文件中参数结构相同。
- 该方法生成的测试用例定义文件命名格式为Test_*optype_001*_ sub_case_*001_format_type，format和type*的取值为该case中第一个Output中的值。若case中没有Output，则文件名不包含后缀，文件名为Test_*OpType_001*_ sub_case_*001*。

2. 将算子测试用例定义文件中需要随机生成的字段的值设为"fuzz"。

   此时显示**K-Level Test Cases**配置项，包含**Fuzz Script Path**和**Fuzz Case Num**选项。

3. 在**Fuzz Script Path**中输入fuzz生成测试参数的脚本的相对路径或绝对路径，如：“fuzz_shape.py”。
4. 在**Fuzz Case Num**中输入利用fuzz脚本生成多少条测试用例，如：2000。
5. （可选）在**Fuzz Function**中配置自定义的随机生成待测试参数的方法，若不配置默认使用fuzz_branch方法。
6. fuzz用例不采取正交生成，因此在该场景下算子测试用例定义文件中各字段的取值需唯一。

步骤5 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。

TBE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。

AI CPU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*/aicpu_kernel”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，以及请勿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}*”或“testcases/st/*OpType*/aicpu_kernel”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 （可选）AI CPU算子引用的三方库路径可通过修改算子工程目录下的“cpukernel/CMakeLists.txt”文件进行配置。

- 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
  )
  ```

- link_libraries：添加编译算子依赖的库文件。

  示例如下：

  ```
  link_libraries(
  libName1
  libName2)
  ```

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 运行ST测试用例。

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（TBE算子：“testcases> st > *OpType* > *{SoC Version}* > xxxx.json”，AI CPU算子：“testcases > st > *OpType* > *aicpu_kernel* > xxxx.json”），选择“Run ST Case 'xxx.json'”。

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st_cases 。

参数	说明
Execute Mode	• Remote Execute：远程执行测试 • Local Execute：本地执行测试 **说明** – Windows操作系统中不支持Local Execute功能。 – 动态shape算子工程运行ST测试时，若编译机器跟运行机器是两台不同的机器，在编译机器和运行机器上都部署算子包，方可支持Remote Execute功能。
Deployment	当Execute Mode选择Remote Execute时，deployment功能，详细请参见**13.3 Ascend Deployment**，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
CANN Machine	CANN工具所在设备deployment信息。 **说明** 该参数仅支持Windows操作系统。
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。 • 也可以单击  ，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*。
Operator Name	选择需要运行的算子。
SoC Version	选择昇腾AI处理器类型。 **说明** 昇腾AI处理器类型为Atlas 推理系列产品、Atlas 训练系列产品及Atlas A2 训练系列产品系列时，请用户根据实际情况选择具体的类型。
Executable File Name	下拉选择需要执行的测试用例定义文件。 若对AI CPU算子进行ST测试，测试用例文件前有（AI CPU）标识。
Toolchain	工具链配置器，根据已安装的CANN包预置架构一致的自定义Toolchain，支持本地和远程编译功能。 可单击“Manage toolchains…”自定义配置Toolchain，配置详情请参见**13.4 Toolchains**。
Case Names	选择运行的Case Name。 默认全选所有用例，可以去除勾选部分不需要运行的用例。
Enable Auto Build&Deploy	选择是否在ST运行过程中执行编译和部署。 默认开启。
Advanced Options	高级选项

参数	说明
ATC Log Level	选择ATC日志级别 ● INFO ● DEBUG ● WARNING ● ERROR ● NULL
Precision Mode	精度模式 ● force_fp16 ● allow_mix_precision ● allow_fp32_to_fp16 ● must_keep_origin_dtype
Device Id	设备ID，设置运行ST测试的芯片ID。用户根据使用的AI芯片ID填写。
Error Threshold	配置自定义精度标准，取值为含两个元素的列表：[val1，val2] ● val1：算子输出结果与标杆数据误差阈值，若误差大于该值则记为误差数据。 ● val2：误差数据在全部数据占比阈值。若误差数据在全部数据占比小于该值，则精度达标，否则精度不达标。 取值范围为[0.0,1.0]。
Generate Error Report	生成比对不一致数据的报表。 该选项默认开启。针对比对失败的用例，将算子期望数据与实际用例执行结果不一致的数据独立生成*{case.name}*_error_report.csv文件，文件所在目录为“testcases/st/out/*OpType*/run/out/test_data/data/st_error_reports”。单个csv文件保存数据的上限为5万行，超过则依次生成新的.csv文件，文件命名如：*{case.name}*_error_report*0*.csv。 使用“Generate error report”功能前，需要指定算子期望数据生成函数自定义脚本，请参见**步骤3**。

参数	说明
Enable System Profier	使能profiling，获取算子在昇腾AI处理器上的性能数据。该选项默认关闭。 该功能需要将运行环境中的msprof工具所在路径配置到PATH环境变量中，msprof工具所在路径为：toolkit工具路径下的toolkit/tools/profiler/bin/msprof。 对于TBE算子，打开情况下会出现下拉框，可下拉选择AI Core性能指标采集项，采集项如下： • PipeUtilization（默认） • ArithmeticUtilization • Memory • MemoryL0 • MemoryUB • ResourceConflictRatio

说明

ST测试支持板端日志级别设置及查询，具体操作请参见13.7 日志管理。

步骤3 Host侧运行用户设置。

Host侧运行用户需在deployment中添加，并且该用户必须为HwHiAiUser属组。配置deployment请参见13.3 Ascend Deployment。

步骤4 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runtime或compiler、driver组件的安装路径。若没有配置，请参考以下操作。

  a. 修改~/.bashrc文件，在文件中添加如下内容。

  ```
  # Ascend-cann-toolkit环境变量（请根据实际路径修改）
  source $HOME/Ascend/ascend-toolkit/set_env.sh
  ```

  b. 使环境变量生效。

  ```
  source ~/.bashrc
  ```

- 在**Environment Variables**中添加环境变量。

  在运行ST测试用例中运行的窗口对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=${ASCEND_HOME}/<target architecture>-linux;    //仅AI CPU算子需要配置
  #若远程设备为推理环境：
  LD_LIBRARY_PATH=${ASCEND_DRIVER_PATH}/lib64:${ASCEND_HOME}/lib64:$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:${ASCEND_HOME}/lib64:$LD_LIBRARY_PATH;
  ```


请根据driver和CANN的实际安装路径配置，远程操作系统的使用架构修改环境变量内容。

- 单击  ，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5 单击“Run”。

MindStudio会根据算子测试用例定义文件在算子根目录“testcases/st/out/<operator name>”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。运行成功后会打印对比报告及保存运行信息。

- 将执行结果和与标杆数据对比报告打印到Output窗口中。
  - 总体信息
    total_count：对比元素总数量。
    max_diff_thd：最大误差阈值，若存在预期结果与真实结果误差超出该阈值，直接判定该测试用例失败。
  - 详细信息
    Index（对比元素序号）、ExpectOut（期望输出）、RealOut（实际输出）、FpDiff（精度误差）、RateDiff（误差比）。
  - 误差容忍信息及结果
    预期结果与真实结果误差小于DiffThd（误差阈值）的数量占比高于PctThd（准确率阈值）则Result（比对结果）为通过，否则失败，同时会打印出PctRlt（实际准确率）。
  - system_profiler信息及结果
    在**步骤2**开启“Enable System Profier”开关，配置采集项后才会以表格形式输出system_profiler信息及结果。具体字段含义请参见《**性能分析工具使用指南**》。
- 在算子根目录“testcases/st/out/<operator name>”下生成st_report.json文件。st_report.json文件信息含义请参见**表8-16**。

表 8-16 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。
	trace_detail	-	运行细节。

字段			说明
		st_case_info	测试信息，包含如下内容。 – 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，包含如下内容。 – status：阶段运行状态，表示运行成功或者失败 – result：输出结果 – stage_name：阶段名称 – cmd：运行命令
	case_name	-	测试名称。
	status	-	测试结果状态，表示运行成功或者失败。
	expect	-	期望的测试结果状态，表示期望运行成功或者失败。
summary	-	-	统计测试用例的结果状态与期望结果状态对比的结果。
	test case count	-	测试用例个数。
	success count	-	测试用例的结果状态与期望结果状态一致的个数。
	failed count	-	测试用例的结果状态与期望结果状态不一致的个数。

----结束

8.12 其他功能及操作

8.12.1 算子工程管理

8.12.1.1 导入算子工程样例

Ascend-cann-toolkit开发套件包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 配置完成Ascend-cann-toolkit开发套件包后，Linux场景下需执行如下操作。

```
cp -r Ascend-cann-toolkit安装目录/ascend-toolkit/latest/tools/msopgen/template/operator_demo_projects $HOME/MindstudioProjects
chmod -R 750 $HOME/MindstudioProjects/operator_demo_projects
```

*$HOME/MindstudioProjects*为用户自定义存放工程的目录。

步骤2 进入工程导入页面。

- MindStudio欢迎界面：在“Welcome to MindStudio”窗口中，单击“Open”。
- MindStudio工程界面：在顶部菜单栏单击“File > Open...” 。

步骤3 在“Open File or Project”窗口中，选择样例工程所在文件夹。

- Windows场景下样例工程所在路径：C:\Users*用户名*\.mindstudio\huawei\adk\remote*CANN包版本号*\toolkit\tools\msopgen\template\operator_demo_projects
- Linux场景下样例工程所在路径：*$HOME/MindstudioProjects*/operator_demo_projects

步骤4 单击“OK”，导入样例工程。

说明

如该工程存在代码风险，在打开时会弹出信任窗口。

- 如该工程源码可被信任且安全，请单击“Trust Project”。（可通过勾选“Trust project in *<工作区目录>*”复选框信任该目录下的所有工程。）
- 如该工程不被信任，仅用于查看其中源码，请单击“Preview in Safe Mode”进入安全模式预览。
- 如放弃打开该工程，请单击“Don't Open”取消工程导入操作。

步骤5 若工作窗口已打开其他工程，会出现确认提示。

- 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。

----结束

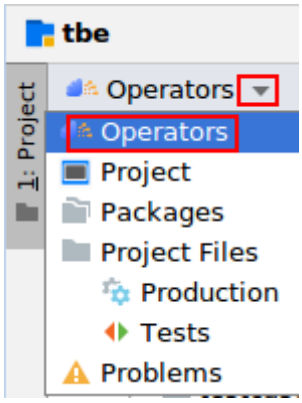
8.12.1.2 切换到算子逻辑视图

开发者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。

开发者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。

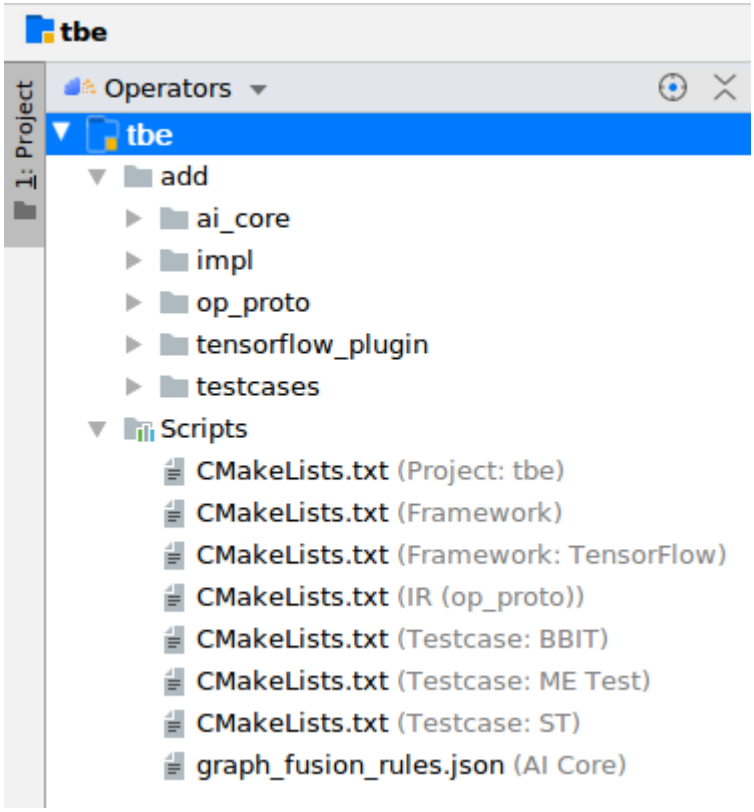
在算子工程的Project窗口中，单击 按钮，在下拉框中选择“Operators”，如图8-20所示。

图 8-20 切换入口



切换成功后，可以查看算子逻辑视图，如图8-21所示。

图 8-21 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和算子测试用例。

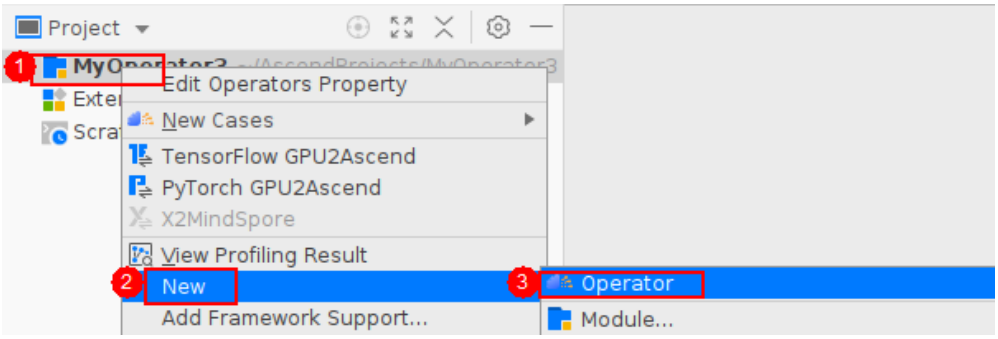
8.12.1.3 新增/删除算子

本节介绍如何在已创建的工程中新增算子和删除已有算子。

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8-22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。

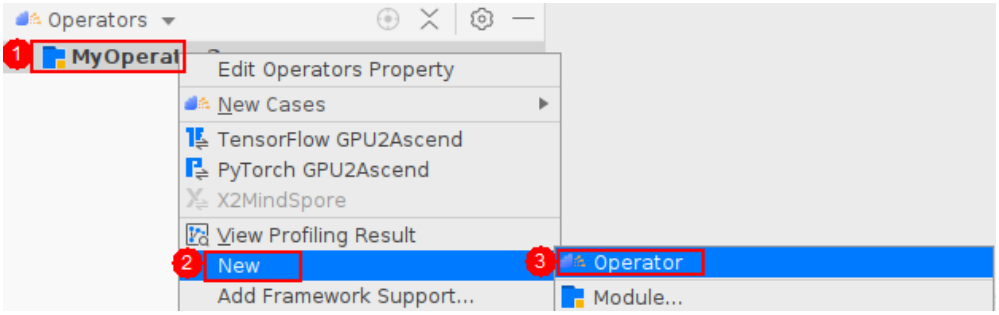
图 8-22 在算子工程视图中新增算子



在算子逻辑视图中新增算子

在算子逻辑视图中，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8-23所示。如果算子类型重复，会提示用户是否进行覆盖。

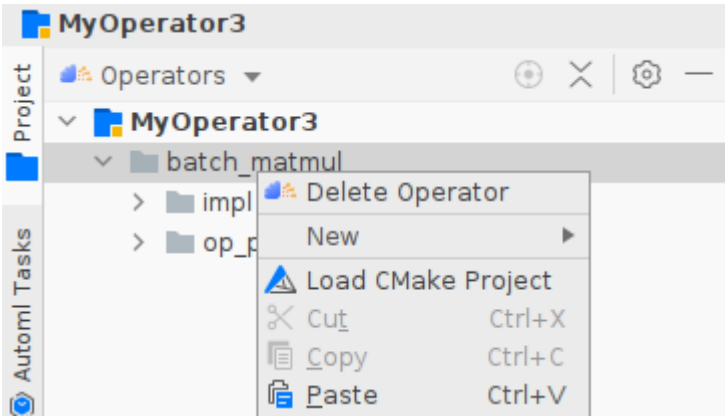
图 8-23 在算子逻辑视图中新增算子



在算子逻辑视图中删除算子

在算子逻辑视图中，选中待删除的算子名称，右键选择“Delete Operator”，如图8-24所示。

图 8-24 在算子逻辑视图中删除算子

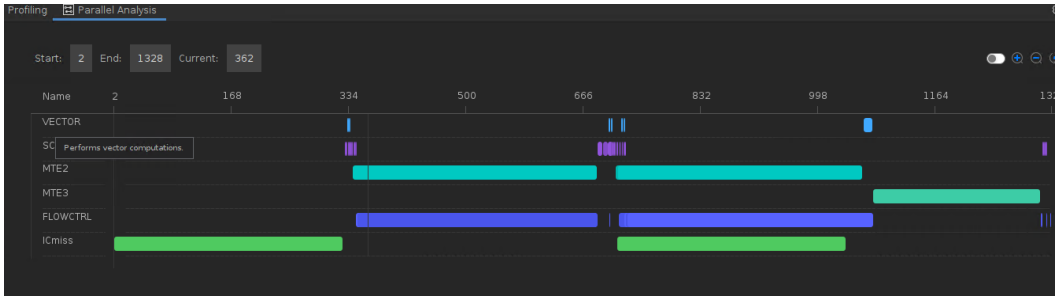


8.12.2 查看单算子 Profiling 数据

查看 Profiling 数据

在执行TBE算子UT测试时，设置运行配置参数Target为Simulator__TMModel，并运行UT测试用例。Profiling执行成功后，在IDE下方控制台会展示执行仿真过程中所生成的Profiling性能数据。如果已经执行过UT测试，通过右键单击算子工程名称，并选择“View Profiling Result”查看Profiling性能数据。Profiling性能数据展示如图8-25所示。

图 8-25 查看 Profiling 性能数据



并行度视图（Parallel Analysis）展示单算子运行中各个指令执行的并行度情况，横轴为时间（Tick，时钟周期），纵轴为各指令单元，鼠标移动到时间块上，可以显示单条指令的消耗时间以及相关指令。

说明

“Start”和“End”表示数据展示的时间范围，“End”为Cycle数，此数值越大，代表运行时间越长。图中数据展示的时间范围可通过滚动鼠标放大或缩小进行变更。

设置代码跳转功能

针对TIK方式开发的算子，如果使用从并行度视图（Parallel Analysis）跳转到TBE、CCE代码功能，需要在运行UT测试用例前，开启Debug开关。

开启Debug开关的方法：在算子代码中的BuildCCE接口中增加config参数，并设置参数tbe_debug_level值为2，编译生成cce代码，生成的算子目标文件中包含debug信息。

TIK算子开启Debug配置参数如下。

```
config = {
        "tbe_debug_level": 2
        }

 self.tik_instance.BuildCCE(
     kernel_name = self.kernel_name,
     inputs=(self.var_gm, self.indices_gm, self.updates_gm),
     enable_l2 = Fales,
     config=config)
```

TIK算子开启Debug开关，并执行UT测试后，显示的并行度视图（Parallel Analysis）如图8-26所示。右键单击时间块，显示“Goto TIK Code”和“Goto CCE Code”，单击即可跳转。

图 8-26 并行度视图（Parallel Analysis）

说明

- 并行度视图（Parallel Analysis）的右上角设置了隐藏TIK Code内部操作的开关，该默认关闭。开启后，当单个TIK Code代码块无法跳转时，在并行度视图上隐藏该代码块。
- “Goto TIK Code”跳转功能暂不支持跳转至对应代码行，需自行查找对应的TIK Code。

8.12.3 TIK 功能调试

简介

MindStudio支持TIK算子的可视化调试，可以实现断点设置、单步调试、连续运行直到结束或下一断点、查看变量信息、退出调试等功能。

前提条件

- 完成环境变量配置，需要完成用户级别的环境变量配置。若没有配置，请参考以下操作。
  a. 修改~/.bashrc文件。在文件中添加如下内容。
  ```
  # Ascend-cann-toolkit环境变量（请根据实际路径修改）
  source $HOME/Ascend/ascend-toolkit/set_env.sh
  ```
  b. 执行命令。
  ```
  source ~/.bashrc
  ```
- 已完成TIK算子实现代码。

操作步骤

本章节以add算子为例介绍TIK功能调试方法。

步骤1 编写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 tbe import tik
from tbe.common.platform import set_current_compile_soc_info

def simple_add():
    tik_instance = tik.Tik(disable_debug=False)
    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__':
    set_current_compile_soc_info（"Ascend*xxx*"）  # 请根据芯片实际版本设置，具体请参考
set_current_compile_soc_info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) # 添加TIK调试代码
    print(model_data)
```

说明

调试TIK Debug时，用户需自行修改tik_instance的描述为tik.Tik(disable_debug=False)，操作方法如下：

在add算子调试代码.py文件中添加如下加粗字体信息。

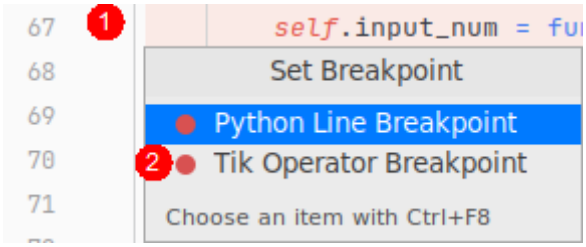
```
self.tik_instance = tik.Tik(disable_debug=False)
```

步骤2 设置断点。

- 添加断点

  单击需要设置tik断点的行，在弹出的对画框中选择“Tik Operator Breakpoint”。如图8-27。

  图 8-27 设置断点

  须知

  - TIK断点必须添加在TIK语句上。
  - 若一行代码被分为多行，需在最后一行设置断点。

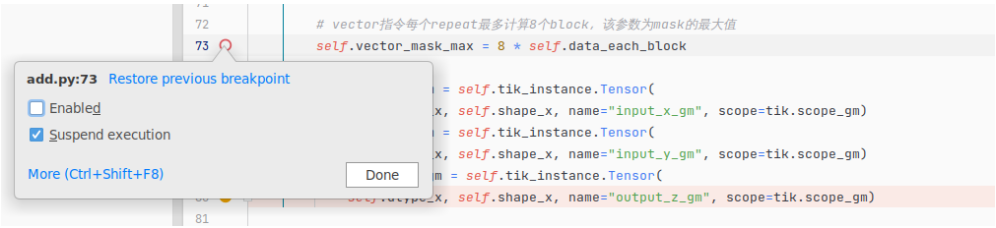
- 删除断点

  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●，在弹出界面中取消勾选“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。

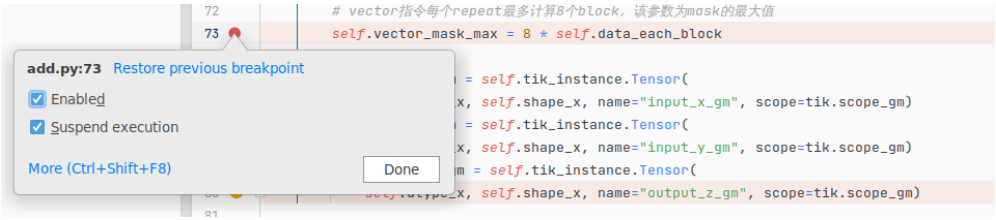
图 8-28 断点 Disable



- 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。

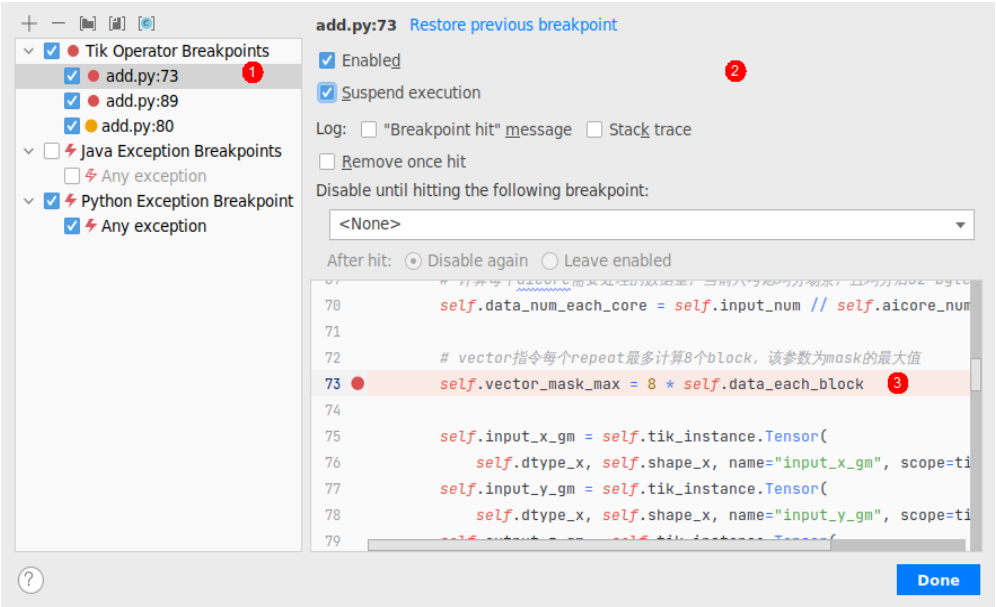
图 8-29 断点 Enable



- 查看断点。通过该功能，用户可以查看设置过的断点。

  单击Disable断点或Enable断点界面的“More(Ctrl+Shift+F8)”，弹出图8-30所示断点查看界面。

图 8-30 TIK 断点查看界面



左侧序号1断点列表展示了当前代码中所设置的所有断点，序号2对应相应断点的停用（Disabled）或启用（Enabled）功能，序号3为代码预览区域。

- 单击左侧断点列表中，断点左侧的复选框，勾选表示该断点启用，相应序号2处“Enabled”功能被勾选，同时在代码预览区域中，对应代码行左侧行号处

显示  图标；取消勾选表示该断点停用，相应序号2处“Enable”功能被取消勾选，同时在代码预览区域中，对应代码行左侧行号处显示  图标。

- 选中断点列表处的断点，单击上方的  可以删除断点，同时代码预览区域，相应代码行号处的断点图标随之删除。
- 在代码预览区域，代码行号处单击，则该行被标记为断点，同时断点列表中会自动增加该断点信息。

步骤3 运行调试。

- 首次调试，右键单击simple_add.py文件选择“Tik Operator Debug”运行调试。
- 非首次调试，可以通过单击工具栏  或“Shift+F9”按键运行调试。

步骤4 调试执行。

调试执行操作步骤请参见**2.3.2.3 调试执行**。

----结束

8.12.4 UT 测试接口参考

8.12.4.1 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“*$HOME*/Ascend/ascend-toolkit/latestcompiler/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。
    不带属性算子case示例：
    ```
    {
      "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"
    }
    ```
    带属性算子case示例：
    ```
    # 以conv2d算子为例，详细算子实现请参见“样例工程”提供的conv2d算子实现文件。
    # def get_op_support_info(inputs, weights, bias, offset_w, outputs, strides, pads,
    dilations,groups=1, data_format='NCHW', offset_x=0, kernel_name="conv2d")

    {
      "params": [
        # inputs参数（tensor）
        {'shape': (1, 1, 16, 16, 16), "format": "NC1HWC0", 'ori_shape': (1, 16, 16, 16), 'ori_format':
    'NCHW', 'dtype': 'float16', "param_type": "input", "value_range": [1.0, 2.0]},
        # weights参数（tensor）
        {'shape': (1, 1, 16, 16), "format":"FRACTAL_Z", 'ori_shape': (16, 16, 1, 1), 'ori_format':
    'NCHW', 'dtype': 'float16', "param_type": "input", "value_range": [1.0, 2.0]},
        # bias参数
        None,
        # offset_w参数
        None,
        # outputs参数（tensor）
        {'shape':(1, 1, 16, 16, 16), 'ori_shape':(1, 16, 16, 16), "format": "NC1HWC0",  'ori_format':
    'NCHW', "param_type": "output", 'dtype': 'float16'},
        # strides参数
        (1, 1, 1, 1),
        # pads参数
        (0, 0, 0, 0),
        # dilations参数
        (1, 1, 1, 1),
        # 可选groups参数
         1,
        # 可选data_format参数
         'NCHW',
        # 可选offset_x参数
         0
      ],
    ```



```
    "case_name": "test_conv2d_case_1",
    "expect": "success"
}
```

该dict中key字段含义如表8-17所示：

表 8-17 key 字段配置信息

参数	值
params	该字段在测试用例运行时透传给算子接口。该字段中的参数应与算子接口的参数顺序一致。 ● 若输入的参数为tensor，可选择如下字段传递。 – shape：tensor的形状 – ori_shape：tensor的原始形状 – format：tensor的格式 – ori_format：tensor的原始格式 – param_type：tensor类型 – dtype：tensor的数据类型 – value_range：tensor取值范围，默认值为[0.1, 1.0] ● 若输入的参数为非tensor，请传递实际参数值。 ● 若输入的参数为空，请传递None。
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“compiler/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：

不带属性算子case示例：

```
{
   "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
}
```

带属性算子case示例：

```
# 以conv2d算子为例，详细算子实现请参见“样例工程”提供的conv2d算子实现文件。
# def get_op_support_info(inputs, weights, bias, offset_w, outputs, strides, pads,
dilations,groups=1, data_format='NCHW', offset_x=0, kernel_name="conv2d")

{
   "params": [
      # inputs参数（tensor）
      {'shape': (1, 1, 16, 16, 16), "format": "NC1HWC0", 'ori_shape': (1, 16, 16, 16), 'ori_format':
'NCHW', 'dtype': 'float16', "param_type": "input", "value_range": [1.0, 2.0]},
      # weights参数（tensor）
      {'shape': (1, 1, 16, 16), "format":"FRACTAL_Z", 'ori_shape': (16, 16, 1, 1), 'ori_format':
'NCHW', 'dtype': 'float16', "param_type": "input", "value_range": [1.0, 2.0]},
      # bias参数
      None,
      # offset_w参数
      None,
      # outputs参数（tensor）
      {'shape':(1, 1, 16, 16, 16), 'ori_shape':(1, 16, 16, 16), "format": "NC1HWC0",  'ori_format':
'NCHW', "param_type": "output", 'dtype': 'float16'},
      # strides参数
      (1, 1, 1, 1),
      # pads参数
      (0, 0, 0, 0),
      # dilations参数
      (1, 1, 1, 1),
      # 可选groups参数
       1,
      # 可选data_format参数
       'NCHW',
      # 可选offset_x参数
       0
   "case_name": "test_conv2d_case_1",
   "calc_expect_func": np_conv2d #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**表8-18**下：

表 8-18 key 字段配置信息

参数	值
params	该字段在测试用例运行时透传给算子接口。该字段中的参数应与算子接口的参数顺序一致。 ● 若输入的参数为tensor，可选择如下字段传递。 – shape：tensor的形状 – ori_shape：tensor的原始形状 – format：tensor的格式 – ori_format：tensor的原始格式 – param_type：tensor类型 – dtype：tensor的数据类型 – value_range：tensor取值范围，默认值为[0.1, 1.0] ● 若输入的参数为非tensor，请传递实际参数值。 ● 若输入的参数为空，请传递None。
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。

参数	值
precision_standard	自定义精度标准，取值为：(rtol, atol, Max_atol)。 ● rtol：相对容忍率 ● atol：绝对容忍率 ● Max_atol：（可选）最大容忍率 **说明** 若不配置此字段，按照如下默认精度与期望数据进行比对： ● 数据类型为float16时：双千分之一，(0.001, 0.001, 0.1)，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。 ● 数据类型为float32时：双万分之一，(0.0001, 0.0001, 0.01)，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。 ● 数据类型为int8或uint8时：(0.001, 1, 1)，即每个数据之间的误差不超过一，误差超过一的数据总和不超过总数据数的千分之一。

- 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/latest/**compiler/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_mode：默认为None，如果仅跑算子编译的测试用例，不需要指定。在添加了精度测试用例的时候，需要指定仿真的模式，使用"ca"或者"pv"。 如果指定"tm"将仅输出打点图，不进行精度比对。
  - simulator_lib_path：仿真库所在的路径。

    该路径结构如下所示：

    ```
    simulator_lib_path/
      Ascend*xxx*/
        lib/
            libpv_model.so
            ...
      Ascend*xxx*/
        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。

8.12.4.2 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“compiler/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/latest/**compiler/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(["Ascend*xxx*", "Ascend*xxx*"], ["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=["Ascend*xxx*", "Ascend*xxx*"], 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=["Ascend*xxx*", "Ascend*xxx*"], 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"
    }]
})
```

8.12.4.3 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

  ElementwiseOpUT.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/latest/**compiler/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

  ElementwiseOpUT.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/latest/**compiler/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"
    }]
})
```

8.12.4.4 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

  ReduceOpUT.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/latest/**compiler/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

  ReduceOpUT.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/latest/**compiler/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]
})
```

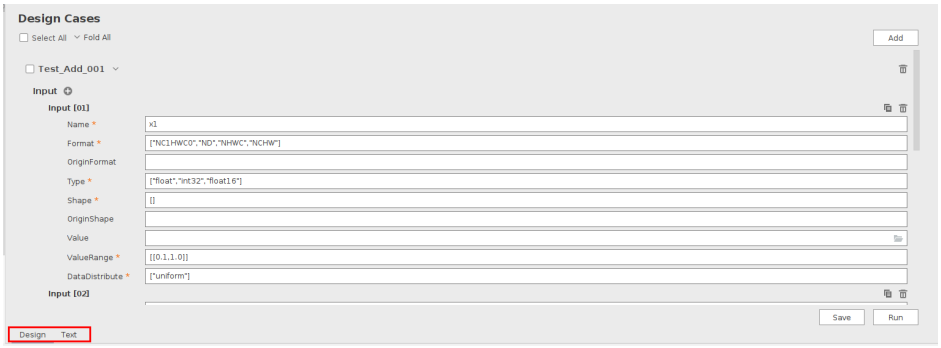
8.12.5 ST 测试其他功能

双击“testcases/st/*OpType*/*{SoC Version}*”或“testcases/st/*OpType*/*aicpu_kernel*”文件夹下的算子测试用例定义文件或者右键该文件选择"*OpType*_case_*timestamp*.json"，会打开ST模板配置界面，该界面中的Design和Text均为json文件的一个视图。配置说明请参见**8.13.3 TBE算子ST测试用例定义文件参数解释**和**8.13.4 AI CPU算子ST测试用例定义文件参数解释**。

其他功能扩展

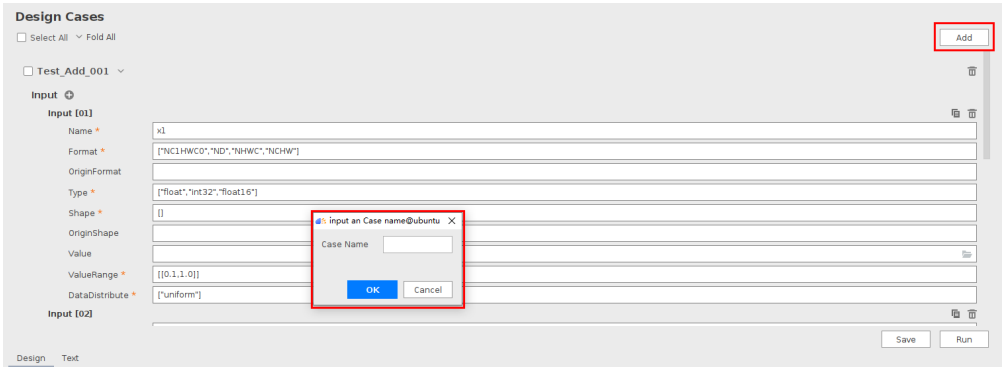
- 支持Design和Text界面视图的切换

  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。



- 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”。

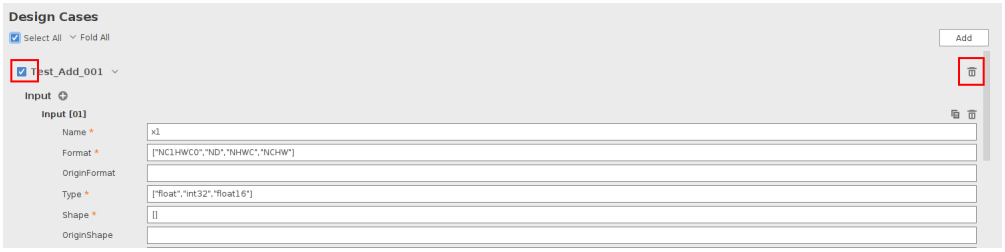


  “Case Name”为由数字、字母、下划线组成的字符串。

  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**8.13.3 TBE算子ST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。

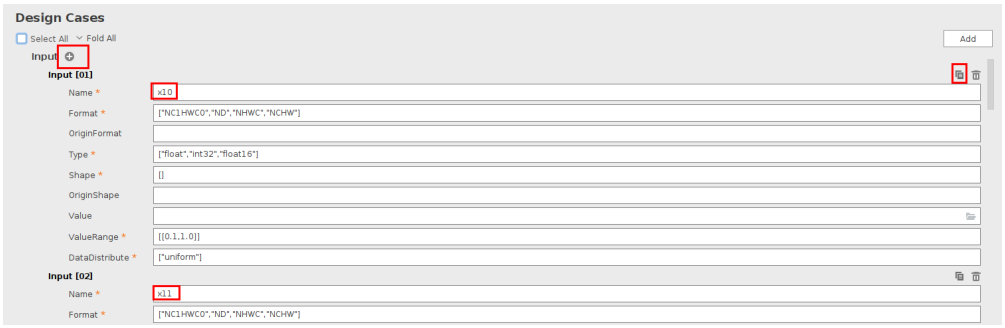


- 复制和新增算子输入

  在算子测试用例定义界面，单击相应Input[xx]右侧的图标，即可复制该Input[xx]成为新的Input[xx]。或者单击新增算子输入。用户根据实际情况进行修改参数项。

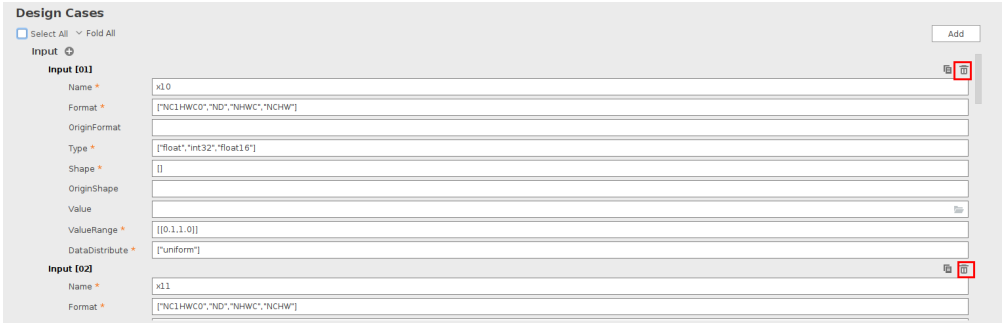
  说明

  如果算子ST工程支持动态多shape算子时，当复制x1的算子输入，此时“Name”参数应该按照x10，x11，x12，...，的格式进行命名，当复制x2的算子输入，此时“Name”参数应该按照x20，x21，x22，...，的格式进行命名。



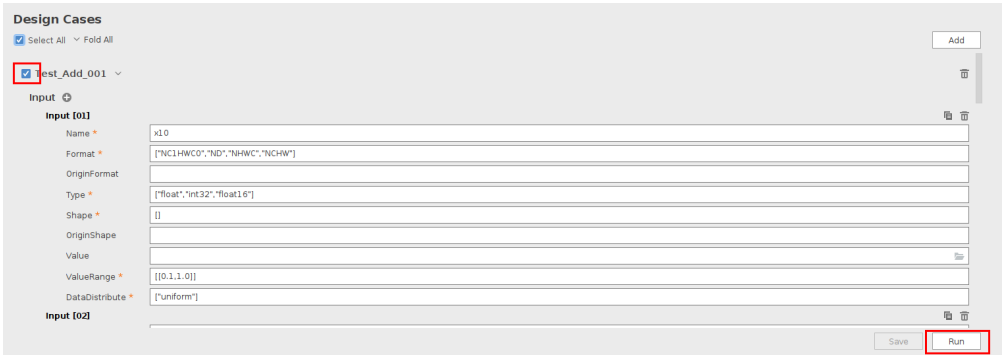
- 删除算子输入

  在算子测试用例定义界面，单击相应Input[xx]右侧的图标，即可删除该Input[xx]。



- 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。



- 支持修改和运行ST测试源码。

  a. 源码生成

  ST测试运行成功后，在./testcases/st/out/*OpName*/src目录中生成ST测试源码。

  b. 源码修改

  用户根据需要可以修改源码内容，实现自定义需求及功能。

  c. 源码运行

  右键单击./testcases/st/out/*<operator name>*，选择“Run St Source”，即可运行修改后的源码。

  说明

  若从其他入口运行ST测试，修改后的源码会被覆盖，请提前备份。

8.13 参考

8.13.1 IR 定义配置说明

- Excel文件配置参数说明

表 8-19 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

- json IR文件配置参数说明
  创建算子工程的json IR文件配置请参见**表8-20**。

表 8-20 json 文件配置参数说明

配置字段		类型	含义	是否必选
Plugin Framework	-	列表	算子所在模型文件的框架类型。 • MindSpore • PyTorch • TensorFlow • Caffe • ONNX	是

配置字段		类型	含义	是否必选
Op Type	-	字符串	算子的Operator Type。 支持输入字母和数字组成，首字母需大写。	是
Input[xx]	-	列表	输入参数描述。	否
	Name	字符串	算子输入参数的名称。	是
	Format	字符串或列表	针对类型为tensor的参数，配置为tensor支持的数据排布格式。如果tensor支持多个数据排布格式，请使用列表。例如["ND","NHWC"]。 包含如下取值： ND,NHWC,NCHW,HWCN,NC1HWC0,FRACTAL_Z等。	否
	Data Type	字符串或列表	输入数据支持的数据类型。 包含如下取值： int32,uint64,float16,float,uint16,int16,int64,uint32,bool,int8,uint8等。 所有输入输出tensor需支持相同数量的数据类型。	是
	Param Type	-	参数类型： • Required • Optional • Dynamic 未配置默认为Required。	是
Output[xx]	-	列表	输出参数描述。	是
	Name	字符串	算子输出参数的名称。	是

配置字段		类型	含义	是否必选
	Format	字符串或列表	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
	Data Type	字符串或列表	输出数据支持的数据类型。 包含如下取值： int32,uint64,float16,float,uint16,int16,int64,uint32,bool,int8,uint8等。 所有输入输出tensor需支持相同数量的数据类型。	是
	Param Type	-	参数类型： ● Required ● Optional ● Dynamic 未配置默认为Required	是
Attribute[xx]	-	列表	属性描述。	否
	Name	字符串	算子属性参数的名称。	是
	Type	字符串或列表	算子参数的类型。 包含如下取值： int、bool、float、string、list_int、list_float等。 若为MindStudio框架包含如下取值： I8_Default,I16_Default,I32_Default,I64_Default,U8_Default,U16_Default,U32_Default,U64_Default,BOOL_Default等	是
	Default Value	-	默认值。	否

配置字段		类型	含义	是否必选
	Param Type	-	参数类型： • Required • Optional 未配置默认为Required。	是

说明

IR Template可以配置多个算子。单击Add按钮增加算子

- 若存在Op Type同名算子，会以后一算子创建算子工程。
- 若Input[xx]或Output[xx]中的Name参数相同，则后一个会覆盖前一参数。
- Input[xx]，Output[xx]中的Data Type与Format需一一对应匹配，如果没有配置Format，自动以“ND”按Type的个数一一对应补齐。

示例：

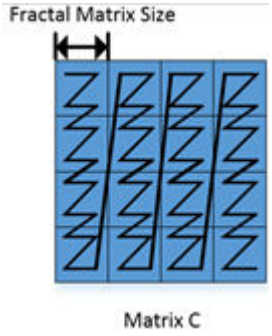
Plugin Framework	-	TensorFlow
Op Type	-	Conv2D
Input[01]	-	-
	Name	x
	Format	["NCHW"]
	Data Type	["fp16"]
	Param Type	Required
Input[02]	-	-
	Name	filter
	Format	["NCHW"]
	Data Type	["fp16"]
	Param Type	Required
Output[01]	-	-
	Name	y
	Format	["NCHW"]
	Data Type	["fp16"]
	Param Type	Required
Attribute[01]	-	-
	Name	strides

	Type	list_int
	Default Value	-
	Param Type	Required
Attribute[02]	-	-
	Name	pads
	Type	list_int
	Default Value	-
	Param Type	Required
Attribute[03]	-	-
	Name	dilations
	Type	list_int
	Default Value	[1,1,1,1]
	Param Type	Optional

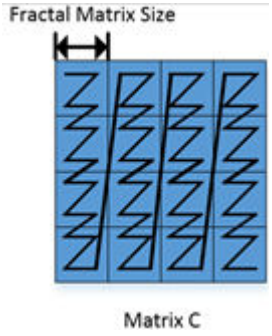
8.13.2 TBE 算子 UT 测试用例定义文件参数解释

表 8-21 TBE 算子 UT 测试用例定义 json 文件

参数		说明
Operator[xx]	-	可选。
-	OpName	算子的名称。
-	TestName	必选。测试用例的名称。
-	SocVersion	必选。支持的昇腾AI处理器类型。 支持输入“all”，表示支持所有的昇腾AI处理器。
-	FunctionType	必选。当前支持的函数类型，支持的参数如下： ● add_precision_case：为当前算子添加一个精度测试用例。 ● add_case：为当前算子添加一个算子编译的测试，测试算子是否可以支持该规格，编译出.o文件。
Input[xx]	-	必选。 算子的输入。
-	Name	可选。算子输入的名称，string类型。

参数		说明
-	Format	必选。 String类型。 输入tensor数据的排布格式，不允许为空。 常见的数据排布格式如下： ● 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： 
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

参数		说明
-	Type	必选。 String类型。 所有输入输出tensor需支持相同数量的数据类型。 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，一维数组。 输入tensor支持的形状，所有输入输出tensor需支持相同数量的形状。 例如： [8, 3, 256, 256]。 若输入非法的形状会报错，例如：[0]。 需要注意，配置的shape需要和format相匹配。
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。
Output[xx]	-	必选。 算子输出。
-	Name	算子输入的名称，string类型。

参数		说明
-	Format	必选。 String类型。 输出tensor数据的排布格式，不允许为空。 常见的数据排布格式如下： ● 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： 
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

参数		说明
-	Type	必选。 String类型。 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。 输入tensor支持的形状。 例如： [8, 3, 256, 256]。 若输入非法的形状会报错，例如：[0]。
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。
Attribute	-	可选。
-	Name	可选。string类型。

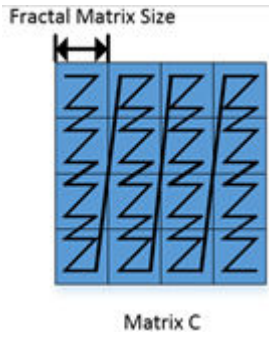
参数		说明
-	Type	若配置attr，则为必选属性支持的类型。 string类型。 输出数据支持的数据类型： ● bool ● int ● float ● string ● list_bool ● list_int ● list_float ● list_string ● list_list_int
-	Value	若配置attr，则为必选，且不允许为null。 string类型。 属性值，根据type的不同，属性值不同。 ● bool: true/false ● int: 10 ● float: 1.0 ● string: “NCHW” ● list_bool: [false, true] ● 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]]

8.13.3 TBE 算子 ST 测试用例定义文件参数解释

表 8-22 TBE 算子 ST 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。

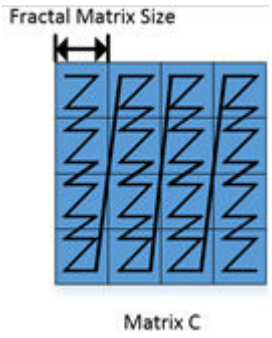
参数		说明
Input[xx]	-	必选。 算子的输入。 **须知** 一个算子所有Input中参数取值的个数都要一致，否则测试用例生成会失败。 例如：Input[01]的format支持的类型个数2，则Input[02]的format支持的类型个数也需要为2。 同理，所有Input[xx]中的type、shape、data_distribute和value_range的取值个数也需要保持一致。
-	Name	必选。 算子输入的名称，string类型。 若配置期望函数，则算子输入的名称需与期望函数中输入的名称保持完全一致。

参数		说明
-	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。 常见的数据排布格式如下： ● 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：  ● RESERVED：预留，当**Format**配置为该值，则Type必须配置为“UNDEFINED”，代表算子的此输入可选。 注意，Format、Type和Shape的数量需保持一致。
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

参数		说明
-	ShapeRange	可选。Shape为动态时，unknow shape的取值范围。默认为[1, -1]。 例如，若Shape取值为[16, -1, 16, -1]，其中的-1表示unknow shape。ShapeRange取值为[[1, 128], [1, -1]]，[1, 128]表示Shape中第一个-1取值范围为1到128，[1, -1]表示Shape中第二个-1的取值范围为1到无穷大。
-	TypicalShape	固定shape，用来生成可用的测试数据。
-	Type	必选。 String或者String的一维数组。 所有输入输出tensor需支持相同数量的数据类型。 输入数据支持的数据类型。 ● bool ● int8 ● uint8 ● int16 ● uint16 ● int32 ● int64 ● uint32 ● uint64 ● float16 ● float ● UNDEFINED：表示输入类型为可选。 注意，Format、Type和Shape的数量需保持一致。
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状，所有输入输出tensor需支持相同数量的形状。 例如： [8, 3, 256, 256]或[[1, 3, 256, 256], [8, 3, 256, 256]]。 若输入非法的形状会报错，例如：[0]。 注意，Format、Type和Shape的数量需保持一致。

参数		说明
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。
-	Value	可选。 路径String或者Tensor数组。 若用户需要指定输入数据时，可通过增加“Value”字段进行配置。有如下两种配置方式： ● 直接输入tensor数据，如tensor的值为[1,2,3,4]。 "Value": [1,2,3,4] ● 输入二进制数据文件的路径，如数据文件为test.bin时。 "Value": "home/HiHwAiUser/projects/test.bin" 二进制数据bin文件需用户自己准备。可以输入绝对路径。 若用户添加了“Value”字段，“DataDistribute”和“ValueRange”字段将会被隐藏。同时需要保证“Format”,"Type", "Shape"字段的值与“Value”数据一致，且每个用例只能测试一种数据类型。
-	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
-	isConst	可选。 bool类型，默认为false。 ● true：若用户需要配置常量输入的用例，则配置该字段，并且其值为true。 ● false：若该字段值为false，则需要配置张量输入用例。
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。
-	Name	必选。 算子输入的名称，string类型。 若配置期望函数，则算子输出的名称与期望函数中输出的名称需保持完全一致。

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。 常见的数据排布格式如下： ● 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：  注意，Format、Type和Shape的数量需保持一致。
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

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 注意，Format、Type和Shape的数量需保持一致。
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状。 例如： [8, 3, 256, 256]或[[1, 3, 256, 256], [8, 3, 256, 256]]。 若输入非法的形状会报错，例如：[0]。 注意，Format、Type和Shape的数量需保持一致。
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。
Attribute	-	可选。
-	Name	若配置attr，则为必选。 string类型。 属性的名称，不为空。

参数		说明
-	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
-	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	可选。 路径String类型。 算子期望数据生成函数路径。
-	Script Function	可选。 String类型。 算子期望数据生成函数名称。
K-Level Test Cases	-	可选。

参数		说明
-	Fuzz Script Path	可选。 路径String类型。 fuzz生成测试参数的脚本路径的绝对路径或者相对路径。
-	Fuzz Function	可选。 String类型。 fuzz脚本中生成fuzz输入的函数名，默认为：“fuzz_branch”。
-	Fuzz Case Num	可选。 int类型。 利用fuzz脚本生成测试用例数量，范围为1-2000。

说明

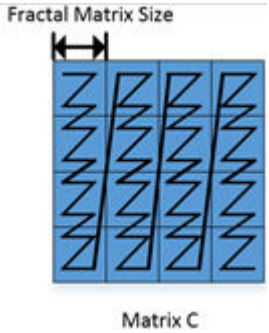
- 当算子信息定义文件（*.ini）中input*x*.paramType=optional时，生成的算子测试用例中input*x*的"format"需配置为"UNDEFINED"或"RESERVED"，"type"为"UNDEFINED"。
- 若配置的**Shape**某个维度为-1或者**Shape**为[-2]时（某维度为-1，如[200, -1] 说明第二个轴长度未知， [-2]场景表示维度未知），生成的算子测试用例中新增**ShapeRange**字段和**TypicalShape**字段。**ShapeRange**字段值为: [[1,-1]]，表示**Shape**字段可以取任何值。用户需要在**TypicalShape**字段给出固定shape值，用于实际测试。
- 算子信息定义文件（*.ini）Tensor的实现format中包含华为自研数据格式，且Tensor的实现format与原始format不同时，用户需在"OriginFormat"和"OriginShape"中填入原始format和shape，将OriginFormat与OriginShape转成离线模型需要的format与shape。
  - "OriginFormat"输入为原始算子支持的数据格式，个数必须与"Format"个数保持一致。
  - "OriginShape"的输入值必须与"Shape"一一对应，且与"Format"和"OriginFormat"格式相匹配。
- Input[xx]、Output[xx]、Attr中除了name和Attr的Type以外的所有参数均支持fuzz输入。

8.13.4 AI CPU 算子 ST 测试用例定义文件参数解释

表 8-23 AI CPU 算子 ST 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。

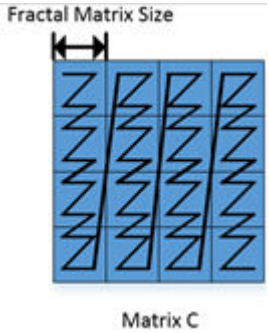
参数		说明
Input[xx]	-	必选。 算子的输入。 **须知** 一个算子所有Input中参数取值的个数都要一致，否则测试用例生成会失败。 例如：Input[01]的format支持的类型个数2，则Input[02]的format支持的类型个数也需要为2。 同理，所有Input[xx]中的type、shape、data_distribute和value_range的取值个数也需要保持一致。
-	Name	必选。 算子输入的名称，string类型。 若配置期望函数，则算子输入的名称需与期望函数中输入的名称保持完全一致。

参数		说明
-	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。 常见的数据排布格式如下： ● 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：  ● RESERVED：预留，当**Format**配置为该值，则type必须配置为“UNDEFINED”，代表算子的此输入可选。 注意，Format、Type和Shape的数量需保持一致。
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

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 ● double ● complex64 ● complex128 ● UNDEFINED：当输入类型为可选时。 注意，Format、Type和Shape的数量需保持一致。
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状。 例如： [8, 3, 256, 256]或[[1, 3, 256, 256], [8, 3, 256, 256]]。 若输入非法的形状会报错，例如：[0]。 注意，Format、Type和Shape的数量需保持一致。
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。

参数		说明
-	Value	可选。 路径String或者Tensor数组。 若用户需要指定输入数据时，可通过增加“Value”字段进行配置。有如下两种配置方式： 直接输入tensor数据，如tensor的值为[1,2,3,4]。 "Value": [1,2,3,4] 输入二进制数据文件的路径，如数据文件为test.bin时。 "Value": "home/HiHwAiUser/projects/test.bin" 二进制数据bin文件需用户自己准备。可以输入绝对路径。若用户添加了“Value”字段，“DataDistribute”和“ValueRange”字段将会被隐藏。同时需要保证“Format”,"Type"，"Shape"字段的值与“Value”数据一致。
-	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
-	isConst	可选。 bool类型，默认为false。 ● true：若用户需要配置常量输入的用例，则配置该字段，并且其值为true。 ● false：若该字段值为false，则需要配置张量输入用例。
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。
-	Name	必选。 算子输入的名称，string类型。 若配置期望函数，则算子输出的名称与期望函数中输出的名称需保持完全一致。

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。 常见的数据排布格式如下： ● 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：  注意，Format、Type和Shape的数量需保持一致。
-	OriginFormat	可选。Tensor的原始format。 不带此字段时，默认Tensor的实现format与原始format一致。

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 ● double ● complex64 ● complex128 注意，Format、Type和Shape的数量需保持一致。
-	Shape	必选。 int类型，一维或者二维数组。 输入tensor支持的形状。 例如： [8, 3, 256, 256]或[[1, 3, 256, 256], [8, 3, 256, 256]]。 若输入非法的形状会报错，例如：[0]。 注意，Format、Type和Shape的数量需保持一致。
-	OriginShape	可选。Tensor的原始shape。 不带此字段时，默认Tensor的实现shape与原始shape一致。
Attribute	-	可选。
-	Name	若配置attr，则为必选。 string类型。 属性的名称，不为空。

参数		说明
-	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
-	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。
K-Level Test Cases	-	可选。
-	Fuzz Script Path	可选。 路径String类型。 fuzz生成测试参数的脚本路径的绝对路径或者相对路径。

参数		说明
-	Fuzz Case Num	可选。 int类型。 利用fuzz脚本生成测试用例数量，范围为1-2000。
-	Fuzz Function	可选。 String类型。 fuzz脚本中生成fuzz输入的函数名，默认为：“fuzz_branch”。

说明

- 当算子信息定义文件（*.ini）中input*x*.paramType=optional时，生成的算子测试用例中input*x*的"format"需配置为"UNDEFINED"或"RESERVED"，"type"为"UNDEFINED"。
- 算子信息定义文件（*.ini）Tensor的实现format中包含华为自研数据格式，且Tensor的实现format与原始format不同时，用户需在"OriginFormat"和"OriginShape"中填入原始format和shape，将OriginFormat与OriginShape转成离线模型需要的format与shape。
  - "OriginFormat"输入为原始算子支持的数据格式，个数必须与"Format"个数保持一致。
  - "OriginShape"的输入值必须与"Shape"一一对应，且与"Format"和"OriginFormat"格式相匹配。
- Input[xx]、Output[xx]、Attr中除了name和Attr的Type以外的所有参数均支持fuzz输入。

9 精度比对

9.1 精度比对简介

自有实现的算子在昇腾AI处理器上的运算结果与业界标准算子（如Caffe、ONNX、TensorFlow）的运算结果可能存在差异：

- 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，这些动作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）运算结果存在差异。
- 用户原始网络可以迁移到昇腾AI处理器上执行训练，网络迁移可能会造成自有实现的算子运算结果与用业界标准算子（如TensorFlow）运算结果存在差异。

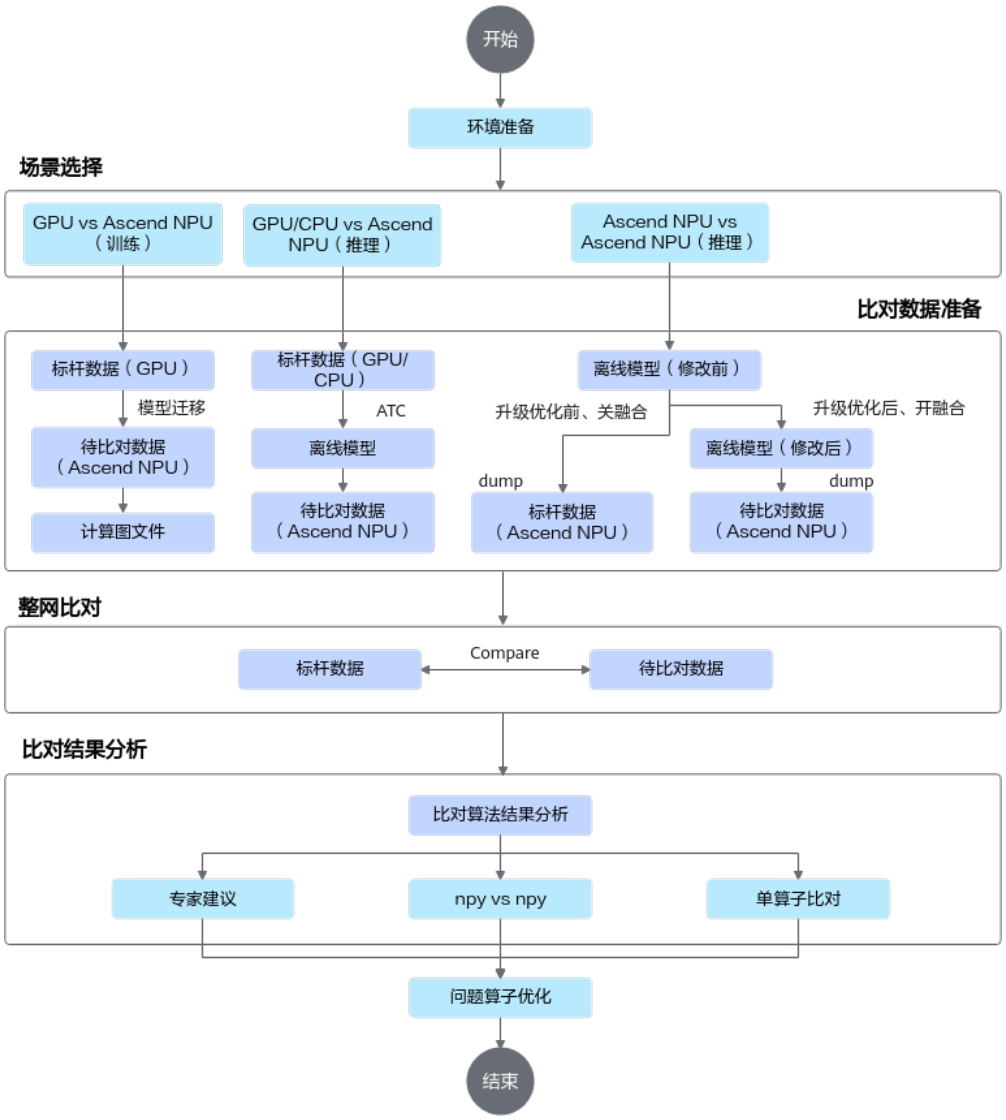
定义：为了帮助开发人员快速解决算子精度问题，需要提供自有实现的算子运算结果与业界标准算子运算结果之间进行精度差异对比的工具。

功能：精度比对工具提供Tensor比对能力，包含余弦相似度、欧氏相对距离、绝对误差（最大绝对误差、平均绝对误差、均方根误差）、相对误差（最大相对误差、平均相对误差、累积相对误差）、KL散度、标准差算法比对维度。

9.2 精度比对流程

精度比对总体流程如下：

图 9-1 精度比对总体流程



1. 环境准备。请参见**9.3 环境准备**。
2. 比对数据准备。请参见**9.5 比对数据准备**。
   a. 推理场景：准备第三方框架原始模型的npy数据文件与离线模型的dump数据文件。
   b. 训练场景：准备基于GPU运行生成的第三方框架原始训练网络npy数据文件与基于昇腾AI处理器运行生成的训练网络dump数据和计算图文件。
3. 整网比对。请参见**9.6 Tensor比对**。
   a. 执行整网比对操作。

      开启MindStudio的“Ascend > Model Accuracy Analyzer”功能，将准备好的比对数据文件配置到对应参数下并配置具体比对参数。
   b. MindStudio执行比对操作并输出比对结果。
   c. 比对结果专家建议（可选）。请参见**9.6.3.3 比对结果专家建议**。
   d. 根据分析结果定位具体问题算子。

e. 执行单算子比对操作。
f. 分析单算子具体问题。

9.3 环境准备

- 请参见《CANN 软件安装指南》手册完成开发或运行环境搭建。
- Python版本支持情况请参见《CANN 软件安装指南》中的“安装开发环境>安装依赖>依赖列表”章节。

9.4 使用约束

- 精度比对功能不支持打开多个工程同时进行比对，可以先完成一个比对程序后再进行下一个。
- 精度比对功能要求protobuf版本在[3.13.0, 3.20.3]范围内。安装示例：**pip3 install protobuf==3.13.0**。非root用户安装，需要在安装命令后加上**--user**，例如：**pip3 install protobuf==3.13.0 --user**。
- 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

9.5 比对数据准备

9.5.1 比对场景

9.6 Tensor比对支持的精度比对场景如表9-1所示，请根据具体场景准备对应的文件。具体文件的获取方式请参见下文具体数据准备章节。

说明

一般情况下，精度比对获取昇腾AI处理器运行生成的dump数据文件与GPU/CPU运行生成的npy数据文件进行比对。若需要进行npy与npy数据文件之间的比对，请参见9.6.6.1 dump数据文件转换为npy数据文件完成dump文件转换成npy文件。

表 9-1 Tensor 比对前数据准备

序号	待比对数据（NPU Dump）	标准数据（Ground Truth）	离线模型文件/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数据文件	非量化原始模型的npy文件(Caffe)	非量化离线模型文件（*.om）
2	非量化离线模型在昇腾AI处理器上运行生成的dump数据文件	非量化原始模型的npy文件(TensorFlow)	非量化离线模型文件（*.om）
3	非量化离线模型在昇腾AI处理器上运行生成的dump数据文件	非量化原始模型的npy文件(ONNX)	非量化离线模型文件（*.om）
4	量化离线模型在昇腾AI处理器上运行生成的dump数据文件	非量化原始模型的npy文件(Caffe)	• 量化离线模型文件（*.om） • 昇腾模型压缩后的量化融合规则文件（.json）
5	量化离线模型在昇腾AI处理器上运行生成的dump数据文件	量化原始模型的npy文件(Caffe)	量化离线模型文件（*.om）
6	量化离线模型在昇腾AI处理器上运行生成的dump数据文件	量化离线模型在昇腾AI处理器上运行生成的dump数据文件	• 不同CANN软件版本、同一模型的不同版本或优化前后的模型数据比对时，不需要指定模型文件和融合规则文件。 • 开启算子融合功能前后分别进行模型转换，生成的dump数据比对时，需要分别指定开启算子融合功能前后的离线模型文件（*.om）或算子映射文件（.json）。
	非量化离线模型在昇腾AI处理器上运行生成的dump数据文件	非量化离线模型在昇腾AI处理器上运行生成的dump数据文件	
训练场景			
1	通过昇腾AI处理器运行生成的训练网络dump数据文件	TensorFlow原始训练网络npy文件	计算图文件（*.txt）
注：上述有关量化、非量化概念请参见《**AMCT工具（Caffe）**》。			

9.5.2 数据格式要求

当前版本支持多种比对方式，因此dump、npy数据文件命名需满足以下要求：

表 9-2 数据文件命名规则

数据类型	数据命名格式	备注
非量化离线模型在昇腾AI处理器上运行生成的dump数据文件	当前包含如下三种文件名格式： ● *{op_type}.{op_name}.{task_id}.{stream_id}.{timestamp}* ● {op_type}.{op_name_lxsliceX}.({stream_id}.){task_id}.{timestamp}.{task_type}.{context_id}.{thread_id}.{device_id} ● {op_type}.{op_name}.({stream_id}.){task_id}.{timestamp}.{task_type}.{context_id}.{thread_id}.{device_id}	命名格式说明： ● op_type（算子类型）、op_name（算子名）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。 ● task_id（任务ID）、stream_id（Stream ID）、output_index（第N个输出）、task_type（任务类型）、context_id（Context ID）、thread_id（线程ID）、device_id（运行卡的Device ID）为0~9数字组成。
量化离线模型在昇腾AI处理器上运行生成的dump数据文件		
npy数据文件（Caffe、TensorFlow或ONNX）	*{op_name}.{output_index}.{timestamp}*.npy	

9.5.3 推理场景数据准备

9.5.3.1 准备 Caffe 模型 npy 数据文件

Caffe 原始数据文件准备要求

Caffe原始npy数据文件要求：

- 文件内容以numpy（.npy）格式保存。
- 文件以*{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，否则无比对结果，原因是精度比对时默认从第一个output_index为0的数据开始。
- 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AMCT再执行去除in-place命令，安装方法请参见《**AMCT工具（Caffe）**》。

进入*Ascend-cann-toolkit安装目录*/ascend-toolkit/latest/**tools/operator_cmp/compare**目录，执行命令去除in-place，命令行举例如下：

```
python3 inplace_layer_process.py -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。

生成 npy 数据文件

本版本不提供Caffe模型numpy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。本文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。

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dump操作，示例代码如下：

```
#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)
            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。

9.5.3.2 准备 TensorFlow 模型 npy 数据文件

生成 npy 数据文件

本版本不提供TensorFlow模型numpy（.npy）数据生成功能，请自行安装TensorFlow环境并提前准备numpy数据。本文仅提供生成numpy格式TensorFlow原始数据“*.npy”文件的样例参考。

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TensorFlow推理脚本，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()]
  ```

  如图9-2所示示例，添加tfdbg的hook。

  图 9-2 Estimator 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：
  ```
  from tensorflow.python import debug as tf_debug
  sess = tf_debug.LocalCLIDebugWrapperSession(sess, ui_type="readline")
  ```

  如图9-3所示示例，在run之前设置tfdbg装饰器。

  图 9-3 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 执行推理脚本。

步骤3 应用程序推理完成后，视图进入调试命令行交互模式**tfdbg**，执行**run**命令。

run命令执行完成后，会在命令行交互界面保存数据为npy格式文件。

----结束

收集 npy 数据文件

run命令执行完成后，需要收集npy文件，但由于**tfdbg**一次只能dump一个tensor，为了自动收集所有npy文件，具体执行操作如下：

步骤1 在**tfdbg**命令行视图下执行**lt > *tensor_name***命令，将所有tensor的名称暂存到文件里。

步骤2 重新开启一个命令行窗口，在新的命令行窗口下执行如下命令，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")}' > tensor_name_cmd.txt
```

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需满足[0-9]{1,255}正则表达式。

步骤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

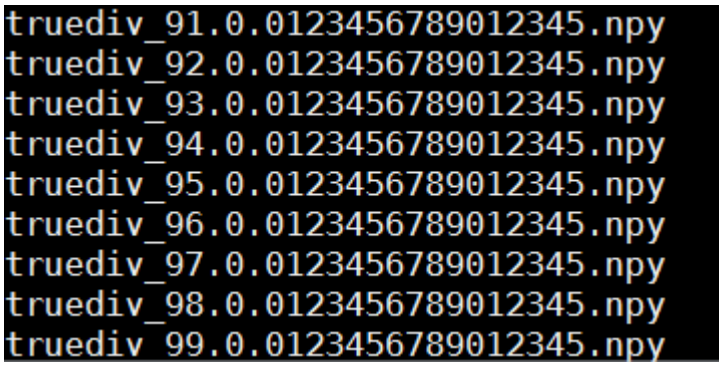
如果命令行界面无法粘贴文件内容，可以在tfdbg命令行中输入“mouse off”指令关闭鼠标模式后再进行粘贴。

步骤4 检查生成的npy文件命名是否符合规则，如图9-4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需满足[0-9]{1,255}正则表达式，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**9.7.2.2 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 9-4 查询.npy 文件



----结束

9.5.3.3 准备 ONNX 模型 npy 数据文件

ONNX 原始数据文件准备要求

- 文件内容以numpy（.npy）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

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。本文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。

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dump操作，示例代码如下：

```
import os
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
node_output_num = [len(node.output) for node in model_onnx.graph.node]
idx = 0
for num, name in zip(node_output_num, node_name):
    for i in range(num):
        data = res[idx]
        file_name = name + "." + str(i) + "." + str(round(time.time() * 1000000)) + ".npy"
        output_dump_path = os.path.join("./onnx_dump/", file_name)
        np.save(output_dump_path, data.astype(np.float16))
        idx += 1
```

说明

需要根据代码中的output_dump_path参数在当前目录新建对应“onnx_dump”目录或自定义目录。

9.5.3.4 准备离线模型 dump 数据

前提条件

本节介绍通过MindStudio提供的dump功能，生成离线模型的dump数据。启动dump前，需要完成以下操作：

- 完成Caffe、TensorFlow或ONNX模型的ATC模型转换，将原始模型转换为OM离线模型。详细介绍请参见**6 模型转换和调优**。
- 完成应用工程的开发、编译和运行，确保具备可执行的应用工程。详细介绍请参见**7 应用开发**。

生成 dump 数据文件

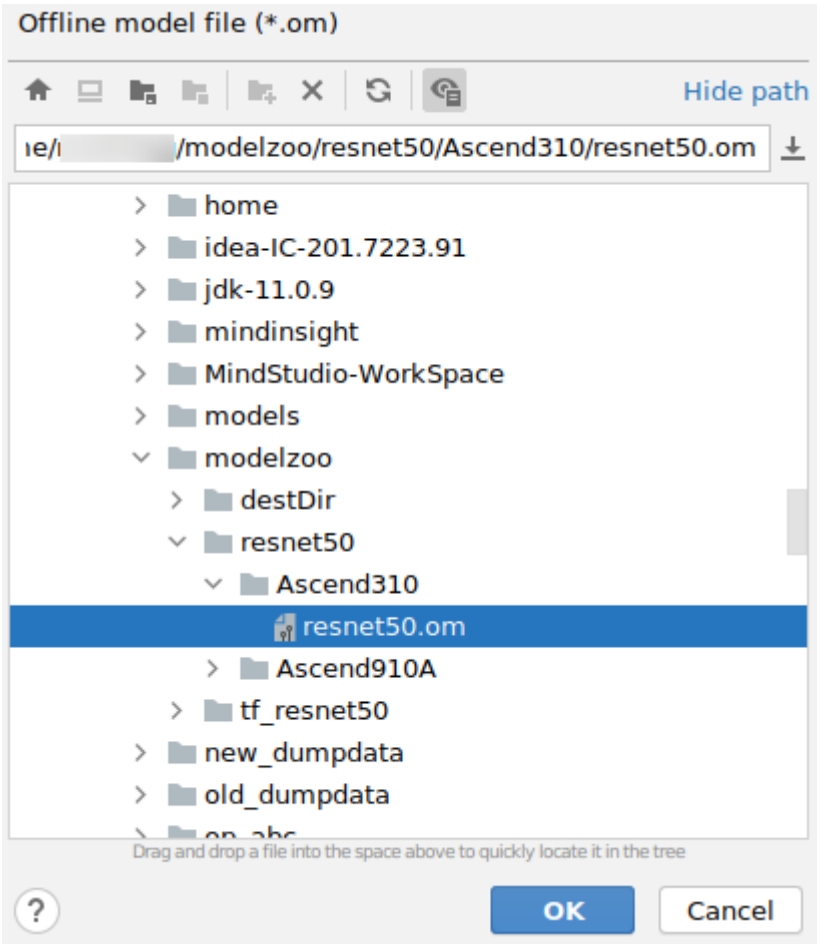
您可以参考以下步骤生成dump数据：

步骤1 打开MindStudio，选择“Ascend > Dump Configuration”菜单，弹出“Select Offline Model”窗口，如**图9-5**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 9-5 选择.om 模型文件



步骤2 选择.om模型文件，单击“OK”，展示模型文件结构，设置dump开关。如图9-6所示。

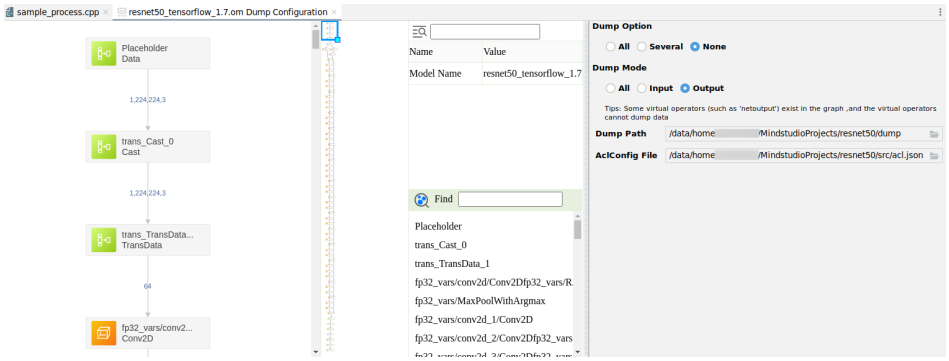
说明

- 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 9-6 设置 dump



通过窗口右侧配置项，设置.om模型文件的dump配置项。

表 9-3 参数说明

参数	说明
Dump Option	配置dump范围。取值为： ● ALL：所有算子开启dump。 ● 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”。 ● None：所有算子不开启dump。
Dump Mode	dump数据模式。取值为： ● All：同时dump算子的输入、输出数据。 ● Input：dump算子的输入数据。 ● Output：dump算子的输出数据。
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。
AclConfig File	Acl配置文件，在dump操作中该文件保存算子的dump配置信息。一般路径为{project_path}/src/acl.json，实际路径可以在{project_path}/.project文件中查找。格式如图9-7或图9-8所示。当选择整个模型dump时，不含layer字段。

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。
- 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。
- 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。
- 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。

图 9-7 整网所有算子开启 dump

```
{
    "dump":{
        "dump_list":[
            {
                "model_name":"ResNet-50"
            }
        ],
        "dump_mode":"input",
        "dump_path":"/home/user/AscendProjects/MyApp/dump"
    }
}
```

图 9-8 部分算子开启 dump

```
{
    "dump":{
        "dump_list":[
            {
                "model_name":"ResNet-50",
                "layer":[
                    "pool1",
                    "res2a_branch2ares2a_branch2a_relu",
                    "conv1conv1_relu"
                ]
            }
        ],
        "dump_mode":"input",
        "dump_path":"/home/user/AscendProjects/MyApp/dump"
    }
}
```

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
```

表 9-4 字段说明

字段	说明
time	dump数据回传落盘时间。格式为：YYYYMMDDhhmmss。
device_id	设备ID。
model_name	模型名称。
model_id	模型ID。
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9.5.2 数据格式要求。
注：如果model_name、op_type、op_name出现了“.”、“/”、“\”、空格时，转换为下划线表示。	

----结束

9.5.4 训练场景数据准备（TensorFlow 1.x）

9.5.4.1 准备基于 GPU 运行生成的 TensorFlow 1.x 原始训练网络 npy 数据文件

前提条件

- 在进行TensorFlow 1.x原始训练网络生成npy或dump数据前，要求有一套完整、可执行的标准TensorFlow模型训练工程。GPU训练环境准备可以参考**在ECS上快速创建GPU训练环境**，链接内容仅供参考，请以实际训练场景为准。
- 不论采用Estimator模式或session.run模式，首先要把脚本中所有的随机全部关闭，包括但不限于对数据集的shuffle，参数的随机初始化，以及某些算子的隐形随机初始化（比如dense算子），确认自己脚本内所有参数均非随机初始化。

生成 npy 数据文件

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TensorFlow训练脚本，添加debug选项设置。

- 如果采用Estimator模式，采用如下方式添加tfdbg的hook。
  a. 新增**from tensorflow.python import debug as tf_debug**导入debug模块。

b. 在生成EstimatorSpec对象实例，即构造网络结构代码位置，新增代码**training_hooks=[tf_debug.LocalCLIDebugHook()]**。

图 9-9 Estimator 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)
```

- 如果采用session.run模式，采用如下方式在run之前设置tfdbg装饰器。

  a. 新增**from tensorflow.python import debug as tf_debug**导入debug模块。

  b. 在session初始化结束后，新增**sess = tf_debug.LocalCLIDebugWrapperSession(sess, ui_type="readline")**。

图 9-10 session.run 模式

```
1  import numpy as np
2  import tensorflow as tf
3  from tensorflow.python import debug as tf_debug
4
5  xs = np.linspace(-0.5, 0.49, 100)
6  x = tf.placeholder(tf.float32, shape=[None], name="x")
7  y = tf.placeholder(tf.float32, shape=[None], name="y")
8  k = tf.Variable([0.0], name="k")
9  y_hat = tf.multiply(k, x, name="y_hat")
10 sse = tf.reduce_sum((y - y_hat) * (y - y_hat), name="sse")
11 train_op = tf.train.GradientDescentOptimizer(learning_rate=0.02).minimize(sse)
12
13 sess = tf.Session()
14 sess.run(tf.global_variables_initializer())
15
16 sess = tf_debug.LocalCLIDebugWrapperSession(sess,ui_type="readline")
17 for _ in range(10):
18         sess.run(y_hat,feed_dict={x:xs,y:10*xs})
19         sess.run(train_op, feed_dict={x: xs, y: 42 * xs})
```

步骤2 执行训练脚本。

步骤3 训练任务停止后，视图进入调试命令行交互模式**tfdbg**，执行**run**命令，训练会往下执行一个step。

```
For more details, see help..
tfdbg> run
```

run命令执行完成后，获取第一个step的训练结果参数，可以依次执行**lt**命令查询已存储的张量，执行**pt**命令查看已存储的张量内容，保存数据为npy格式文件。

----结束

收集 npy 数据文件

run命令执行完成后，需要收集npy文件，但由于**tfdbg**一次只能dump一个tensor，为了自动收集所有npy文件，具体执行操作如下：

步骤1 执行**lt > *gpu_dump***命令将所有tensor的名称暂存到自定义名称的*gpu_dump*文件里。命令行中会有如下回显。

```
Wrote output to tensor_name
```

步骤2 重新开启一个命令行窗口，在新的命令行窗口进入*gpu_dump*文件所在目录（默认在训练脚本所在目录），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 gpu_dump | awk '{print "pt",$4,$4}' | awk '{gsub("/", "_", $3);gsub(":", ".", $3);print($1,$2,"-n 0 -w "$3""'$timestamp'".npy")}'
```

步骤3 复制所有生成的存储tensor的命令（所有以“pt”开头的命令），回到**tfdbg**命令行视图所在窗口，粘贴执行，即可存储所有npy文件。存储路径为训练脚本所在目录。

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

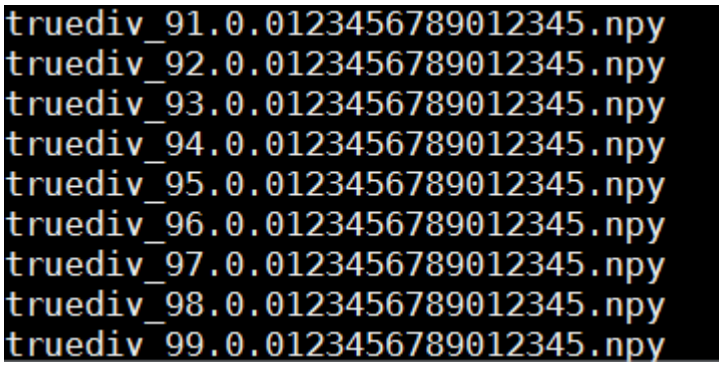
如果命令行界面无法粘贴文件内容，可以在tfdbg命令行中输入“mouse off”指令关闭鼠标模式后再进行粘贴。

步骤4 检查生成的npy文件命名是否符合**9.5.2 数据格式要求**，如**图9-11**所示。

说明

- 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，请参见**9.7.2.2 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 9-11 查询.npy 文件



----结束

9.5.4.2 准备基于昇腾 AI 处理器运行生成的训练网络 dump 数据和计算图文件

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。
- 确保你的代码在网络结构上、算子、优化器的选择上，以及参数的初始化策略等方面跟GPU上训练的代码完全一致，否则比对无意义。
- 不要在一个训练脚本中既做训练又做验证，也就是不要把train和evaluate放到同一个脚本中，否则会生成两组dump数据，导致混淆。
- 目前仅支持AI CPU、AI Core和HCCL算子进行dump数据。

dump 参数配置

步骤1 为了让训练脚本能够dump出计算图，我们在训练脚本中的包引用区域引入os，并在构建模型前设置DUMP_GE_GRAPH参数。

```
import os
...
def main():
    ...
    os.environ['DUMP_GE_GRAPH'] = '2'
```

在训练过程中，计算图文件会保存在训练脚本所在目录中。

步骤2 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）。

```
from npu_bridge.estimator.npu.npu_config import DumpConfig

# 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")
```



```
config = NPURunConfig(
  dump_config=dump_config,
  session_config=session_config
  )
```

关于DumpConfig类的构造函数中每个字段的详细解释，请参见《**TensorFlow 1.15网络模型迁移和训练指南**》手册。

- session.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

custom_op.parameter_map["enable_dump"].b = True
custom_op.parameter_map["dump_path"].s = tf.compat.as_bytes("/home/HwHiAiUser/output")
custom_op.parameter_map["dump_step"].s = tf.compat.as_bytes("0|5|10")
custom_op.parameter_map["dump_mode"].s = tf.compat.as_bytes("all")
custom_op.parameter_map["dump_data"].s = tf.compat.as_bytes("stats")
custom_op.parameter_map["dump_layer"].s = tf.compat.as_bytes("nodename1 nodename2 nodename3")
config.graph_options.rewrite_options.remapping = RewriterConfig.OFF

with tf.Session(config=config) as sess:
  print(sess.run(cost))
```

表 9-5 参数详细说明

参数名	描述
enable_dump	是否开启dump功能。取值为： – True：开启dump功能，从dump_path读取dump文件保存路径。 – False：关闭dump功能。默认值。
dump_path	dump文件保存路径。enable_dump为True时，该参数必须配置。 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，支持配置绝对路径或相对路径（相对执行命令行时的当前路径）。 – 绝对路径配置以“/”开头，例如：/home/HwHiAiUser/output。 – 相对路径配置直接以目录名开始，例如：output。
dump_step	指定采集哪些迭代的dump数据。默认值：None，表示所有迭代都会产生dump数据。 多个迭代用“|”分割，例如：0|5|10；也可以用"-"指定迭代范围，例如：0|3-5|10。

参数名	描述
dump_mode	dump模式，用于指定dump算子输入还是输出数据。取值为： – input：仅dump算子输入数据。 – output：仅dump算子输出数据。默认值。 – all：dump算子输入和输出数据。
dump_data	指定算子dump内容类型，取值为： – tensor：dump算子数据。默认值。 – stats：dump算子统计数据，结果文件为csv格式。 大规模训练场景下，通常dump数据量太大并且耗时长，可以先dump所有算子的统计数据，根据统计数据识别可能异常的算子，然后再指定dump异常算子的input或output数据。
dump_layer	指定需要dump的算子。取值为算子名，多个算子名之间使用空格分隔。

----结束

生成 dump 数据文件

步骤1 执行训练脚本，生成dump数据文件和计算图文件。

- 计算图文件：以“ge”开头的文件，是设置“DUMP_GE_GRAPH=2”生成的计算图文件，存储在训练脚本所在目录。
- dump数据文件：生成在dump_path指定的目录下，即{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“/home/HwHiAiUser/output/20200808163566/0/ge_default_20200808163719_121/11/0”目录下。

表 9-6 dump 数据文件路径格式说明

路径key	说明	备注
dump_path	步骤2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	设备ID。	-

路径key	说明	备注
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。 如果model_name出现了“.”、“/”、“\”、空格时，转换为下划线表示。
model_id	子图ID。	-
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。

说明

- dump文件命名格式需满足9.5.2 数据格式要求。如果文件名称长度超过了OS文件名称长度限制（一般是255个字符），则会将该dump文件重命名为一串随机数字，映射关系可查看同目录下的mapping.csv。
- 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数据量随之增大，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。
- 在docker内执行时，生成的数据存在docker里。

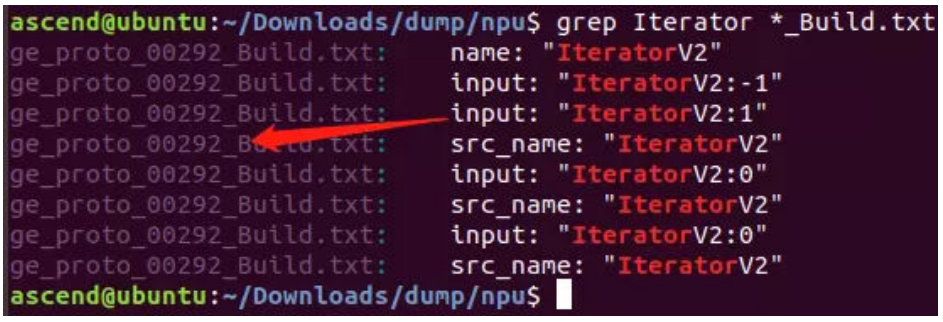
步骤2 选取计算图文件。

说明

由于“ge”开头的dump图文件较多，且dump数据文件中的model_name层可能存在多个文件夹，实际仅需要找到计算图文件，且仅需要model_name为计算图名称的文件夹。以下提供两种方法帮助用户快速找到对应的文件。

- 方法一：在所有以“_Build.txt”为结尾的dump图文件中，查找“Iterator”关键词。记住查找出的计算图文件名称，用于后续精度比对。

```
grep Iterator *_Build.txt
```



如上图所示，“ge_proto_00292_Build.txt”即为需要的计算图文件。

- 方法二：将TensorFlow模型保存为pb文件，然后查看该模型，选取其中一个计算类算子的名字作为关键字，找包含该关键字的计算图文件。计算图名称取计算图文件**graph**下的**name**字段值。

步骤3 选取dump数据文件。

1. 打开**步骤2**中找到的计算图文件，记录第一个graph中的name字段值。如下示例中，记录“ge_default_20201209083353_71”。

```
graph {
  name: "ge_default_20201209083353_71"
  op {
    name: "atomic_addr_clean0_71"
    type: "AtomicAddrClean"
    attr {
      key: "_fe_imply_type"
      value {
        i: 6
      }
}
```

2. 进入以时间戳命名的dump文件存放路径下，我们会看到该目录下存在几个文件夹：

```
ascend@ubuntu:~/Downloads/dump/npu/20201209083219$ cd 0/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0$ ll
total 20
drwxr-xr-x 5 ascend ascend 4096 Dec  9 06:05 ./
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ../
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083219_1/
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083226_21/
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083353_71/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0$
```

3. 找到前面记录的名称为name值的文件夹，例如ge_default_20201209083353_71，这些文件即为需要的dump数据文件。

```
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0$ cd ge_default_20201209083353_71
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71$ ll
total 12
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ./
drwxr-xr-x 5 ascend ascend 4096 Dec  9 06:05 ../
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 6/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71$ cd 6/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71/6$ ll
total 12
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ./
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ../
drwxr-xr-x 2 ascend ascend 4096 Dec  9 06:05 0/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71/6$ cd 0/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71/6/0$ ll
total 8508
drwxr-xr-x 2 ascend ascend    4096 Dec  9 06:05 ./
drwxr-xr-x 3 ascend ascend    4096 Dec  9 06:05 ../
-rwxr-xr-x 1 ascend ascend    3153 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_1_bias_ApplyGradientDescent.23.1607502805243033*
-rwxr-xr-x 1 ascend ascend  786522 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_1_kernel_ApplyGradientDescent.26.1607502805250478*
-rwxr-xr-x 1 ascend ascend     195 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_2_bias_ApplyGradientDescent.18.1607502805233840*
-rwxr-xr-x 1 ascend ascend   30804 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_2_kernel_ApplyGradientDescent.21.1607502805239107*
-rwxr-xr-x 1 ascend ascend    3153 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_bias_ApplyGradientDescent.28.1607502805255216*
-rwxr-xr-x 1 ascend ascend 2408538 Dec  9 06:05 ApplyGradientDescent.GradientDescent_update_dense_kernel_ApplyGradientDescent.30.1607502805261549*
-rwxr-xr-x 1 ascend ascend      76 Dec  9 06:05 AssignAdd.GradientDescent.31.1607502805270321*
-rwxr-xr-x 1 ascend ascend  132151 Dec  9 06:05 BiasAddGrad.gradients_dense_1_BiasAdd_grad_BiasAddGrad.22.1607502805240977*
-rwxr-xr-x 1 ascend ascend    5211 Dec  9 06:05 BiasAddGrad.gradients_dense_2_BiasAdd_grad_BiasAddGrad.17.1607502805233212*
-rwxr-xr-x 1 ascend ascend  132151 Dec  9 06:05 BiasAddGrad.gradients_dense_BiasAdd_grad_BiasAddGrad.27.1607502805254230*
-rwxr-xr-x 1 ascend ascend     692 Dec  9 06:05 Cast.Cast.8.1607502805211837*
-rwxr-xr-x 1 ascend ascend   10294 Dec  9 06:05 Cast.trans_Cast_0.12.1607502805222215*
-rwxr-xr-x 1 ascend ascend  401591 Dec  9 06:05 FrameworkOp.IteratorV2.2.1607502805208598*
-rwxr-xr-x 1 ascend ascend    6100 Dec  9 06:05 GatherV2D.SparseSoftmaxCrossEntropyWithLogits_SparseSoftmaxCrossEntropyWithLogits_GatherV2.10.1607502805218725*
-rwxr-xr-x 1 ascend ascend  525424 Dec  9 06:05 MatMulV2.dense_1_MatMul.11.1607502805220599*
-rwxr-xr-x 1 ascend ascend  146578 Dec  9 06:05 MatMulV2.dense_2_MatMul.13.1607502805226835*
-rwxr-xr-x 1 ascend ascend 1336430 Dec  9 06:05 MatMulV2.dense_MatMul.9.1607502805213454*
-rwxr-xr-x 1 ascend ascend  524364 Dec  9 06:05 MatMulV2.gradients_dense_1_MatMul_grad_MatMul_1.25.1607502805248390*
-rwxr-xr-x 1 ascend ascend  524364 Dec  9 06:05 MatMulV2.gradients_dense_1_MatMul_grad_MatMul.24.1607502805244931*
-rwxr-xr-x 1 ascend ascend  146504 Dec  9 06:05 MatMulV2.gradients_dense_2_MatMul_grad_MatMul_1.20.1607502805236947*
-rwxr-xr-x 1 ascend ascend  146504 Dec  9 06:05 MatMulV2.gradients_dense_2_MatMul_grad_MatMul.19.1607502805235792*
-rwxr-xr-x 1 ascend ascend 1335372 Dec  9 06:05 MatMulV2.gradients_dense_MatMul_grad_MatMul_1.29.1607502805256455*
-rwxr-xr-x 1 ascend ascend   10822 Dec  9 06:05 Mul.gradients_SparseSoftmaxCrossEntropyWithLogits_SparseSoftmaxCrossEntropyWithLogits_grad_mul.16.1607502805231485*
-rwxr-xr-x 1 ascend ascend     524 Dec  9 06:05 OneHotD.SparseSoftmaxCrossEntropyWithLogits_SparseSoftmaxCrossEntropyWithLogits_OneHot.4.1607502805207307*
-rwxr-xr-x 1 ascend ascend     573 Dec  9 06:05 ReduceMeanD.Mean.15.1607502805229752*
-rwxr-xr-x 1 ascend ascend   16113 Dec  9 06:05 SoftmaxCrossEntropyWithLogits.SparseSoftmaxCrossEntropyWithLogits_SparseSoftmaxCrossEntropyWithLogits.14.1607502805227956*
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0/ge_default_20201209083353_71/6/0$
```

----结束

9.5.5 训练场景数据准备（TensorFlow 2.x）

9.5.5.1 准备基于 GPU 运行生成的 TensorFlow 2.x 原始训练网络 npy 数据文件

前提条件

- 在进行TensorFlow 2.x原始训练网络生成npy或dump数据前，要求有一套完整、可执行的标准TensorFlow模型训练工程。GPU训练环境准备可以参考**在ECS上快速创建GPU训练环境**，链接内容仅供参考，请以实际训练场景为准。

- 参见**tfdbg_ascend工具的readme文档**安装TensorFlow 2.x的debug工具tfdbg_ascend。
- 首先要把脚本中所有的随机全部关闭，包括但不限于对数据集的shuffle，参数的随机初始化，以及某些算子的隐形随机初始化（比如dense算子），确认自己脚本内所有参数均非随机初始化。

生成 npy 数据文件

利用TensorFlow的debug工具tfdbg_ascend生成npy文件。详细的操作方法如下：

步骤1 修改TensorFlow训练脚本，在调起模型部分的训练脚本.py文件中修改配置。示例代码如下。

样例一：

1. 导入debug插件。
```
import tfdbg_ascend as dbg
```
2. 在每个step训练启动代码前配置如下代码，例如dump第5个step的数据。
```
    tfdbg.disable()
    if current_step == 5:
      tfdbg.enable()
      tfdbg.set_dump_path('home/test/gpu_dump')
```

样例二：

1. 导入debug插件。
```
import tfdbg_ascend as dbg
```
2. 例如dump第4个step的数据。dbg.enable不配置时，dump功能默认开启；dump路径不指定时，dump文件默认保存在训练脚本所在路径下。
```
class DumpConfig(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
    def on_batch_begin(self, batch, logs={}):
        if batch == 4:
            dbg.enable()
            dbg.set_dump_path("/user/name1/pip_pkg/dump4")
        else:
            dbg.disable()
```
3. 注册回调函数（define callbacks ）。
```
# define callbacks
        callbacks = [
            ModelCheckpoint(
                f'models/model_epochs-{epochs}_batch-{batch_size}_loss-
{loss_function}_{Mask2FaceModel.get_datetime_string()}.h5'),
            LossHistory(batch_size),
            DumpConfig()
        ]

# fit the model 调起模型的代码位置
history = self.model.fit(train_dataset, validation_data=valid_dataset, epochs=1, callbacks=callbacks,
verbose=2)
```

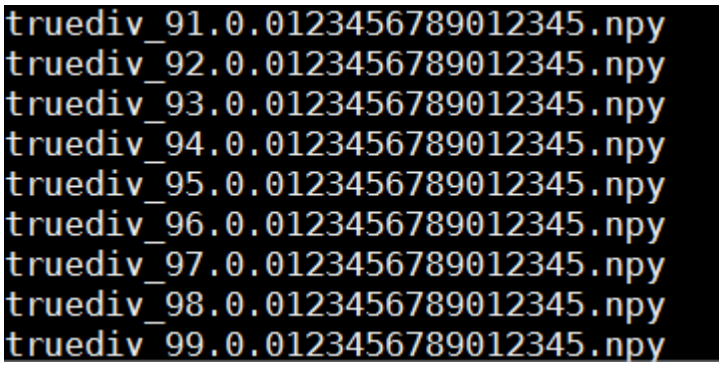
步骤2 执行训练脚本，训练任务停止后，在指定目录下生成*.npy文件。

步骤3 检查生成的npy文件命名是否符合**9.5.2 数据格式要求**，如**图9-12**所示。

说明

- 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**9.7.2.2 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 9-12 查询.npy 文件



----结束

9.5.5.2 准备基于昇腾 AI 处理器运行生成的训练网络 dump 数据和计算图文件

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。
- 确保你的代码在网络结构上、算子、优化器的选择上，以及参数的初始化策略等方面跟GPU上训练的代码完全一致，否则比对无意义。
- 不要在一个训练脚本中既做训练又做验证，也就是不要把train和evaluate放到同一个脚本中，否则会生成两组dump数据，导致混淆。
- 目前仅支持AI CPU、AI Core和HCCL算子进行dump数据。

dump 参数配置

步骤1 为了让训练脚本能够dump出计算图，我们在训练脚本中的包引用区域引入os，并在构建模型前设置DUMP_GE_GRAPH参数。

```
import os
...
def main():
    ...
    os.environ['DUMP_GE_GRAPH'] = '2'
```

在训练过程中，计算图文件会保存在训练脚本所在目录中。

步骤2 修改训练脚本，开启dump功能。在相应代码中，增加如下信息。

```
import npu_device

# 在初始化的时候添加以下配置
```

```
npu_device.global_options().dump_config.enable_dump = True
npu_device.global_options().dump_config.dump_path = "/home/HwHiAiUser/output"
npu_device.global_options().dump_config.dump_step = "0|5|10"
npu_device.global_options().dump_config.dump_mode = "all"
npu.device.global_options().dump_config.dump_data = "stats"
npu.device.global_options().dump_config.dump_layer = "nodename1 nodename2 nodename3"
```

表 9-7 参数详细说明

参数名	描述
dump_config.enable_dump	是否开启dump功能。取值为： ● True：开启dump功能，从dump_path读取dump文件保存路径。 ● False：关闭dump功能。默认值。
dump_config.dump_path	dump文件保存路径。enable_dump为True时，该参数必须配置。 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，支持配置绝对路径或相对路径（相对执行命令行时的当前路径）。
dump_config.dump_step	指定采集哪些迭代的dump数据。默认值：None，表示所有迭代都会产生dump数据。 多个迭代用“|”分割，例如：0|5|10；也可以用"-"指定迭代范围，例如：0|3-5|10。
dump_config.dump_mode	dump模式，用于指定dump算子输入还是输出数据。取值为： ● input：仅dump算子输入数据 ● output：仅dump算子输出数据。默认值。 ● all：dump算子输入和输出数据
dump_config.dump_data	指定算子dump内容类型。取值为： ● tensor：dump算子数据。默认值。 ● stats：dump算子统计数据，结果文件为csv格式。 大规模训练场景下，通常dump数据量太大并且耗时长，可以先dump所有算子的统计数据，根据统计数据识别可能异常的算子，然后再指定dump异常算子的input或output数据。
dump_config.dump_layer	指定需要dump的算子。取值为算子名，多个算子名之间使用空格分隔。

----结束

生成 dump 数据文件

步骤1 执行训练脚本，生成dump数据文件和计算图文件。

- 计算图文件：以“ge”开头的文件，是设置“DUMP_GE_GRAPH=2”生成的计算图文件，存储在训练脚本所在目录。

- dump数据文件：生成在dump_path指定的目录下，即{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“/home/HwHiAiUser/output/20200808163566/0/ge_default_20200808163719_121/11/0”目录下。

表 9-8 dump 数据文件路径格式说明

路径key	说明	备注
dump_path	步骤2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	设备ID。	-
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。 如果model_name出现了“.”、“/”、“\”、空格时，转换为下划线表示。
model_id	子图ID。	-
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。

说明

- dump文件命名格式需满足9.5.2 数据格式要求。如果文件名称长度超过了OS文件名称长度限制（一般是255个字符），则会将该dump文件重命名为一串随机数字，映射关系可查看同目录下的mapping.csv。
- 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数据量随之增大，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。
- 在docker内执行时，生成的数据存在docker里。

步骤2 选取计算图文件。

说明

由于“ge”开头的dump图文件数量较大，且dump数据文件中的model_name层可能存在多个文件夹，实际仅需要找到计算图文件，且仅需要model_name为计算图名称的文件夹。以下提供两种方法帮助用户快速找到对应的文件。

- 方法一：在所有以“_Build.txt”为结尾的dump图文件中，查找“Iterator”这个关键词。记住查找出的计算图文件名称，用于后续精度比对。

```
grep Iterator *_Build.txt
```



```
ascend@ubuntu:~/Downloads/dump/npu$ grep Iterator *_Build.txt
ge_proto_00292_Build.txt:    name: "IteratorV2"
ge_proto_00292_Build.txt:    input: "IteratorV2:-1"
ge_proto_00292_Build.txt:    input: "IteratorV2:1"
ge_proto_00292_Build.txt:    src_name: "IteratorV2"
ge_proto_00292_Build.txt:    input: "IteratorV2:0"
ge_proto_00292_Build.txt:    src_name: "IteratorV2"
ge_proto_00292_Build.txt:    input: "IteratorV2:0"
ge_proto_00292_Build.txt:    src_name: "IteratorV2"
ascend@ubuntu:~/Downloads/dump/npu$
```

如上图所示，“ge_proto_00292_Build.txt”文件即是我们需要找到的计算图文件。

- 方法二：将TensorFlow模型保存为pb文件，然后查看该模型，选取其中一个计算类算子的名字作为关键字，找包含该关键字的计算图文件。计算图名称取计算图文件**graph**下的**name**字段值。

步骤3 选取dump数据文件。

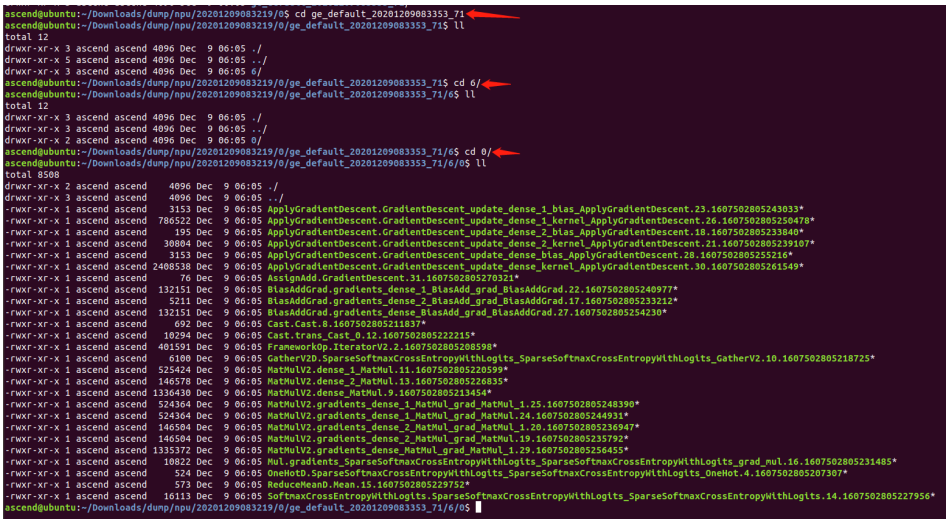
1. 打开**步骤2**中找到的计算图文件，记录第一个graph中的name字段值。如下示例中，记录“ge_default_20201209083353_71”。

```
graph {
  name: "ge_default_20201209083353_71"
  op {
    name: "atomic_addr_clean0_71"
    type: "AtomicAddrClean"
    attr {
      key: "_fe_imply_type"
      value {
        i: 6
      }
}
```

2. 进入以时间戳命名的dump文件存放路径下，我们会看到该目录下存在几个文件夹：

```
ascend@ubuntu:~/Downloads/dump/npu/20201209083219$ cd 0/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0$ ll
total 20
drwxr-xr-x 5 ascend ascend 4096 Dec  9 06:05 ./
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ../
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083219_1/
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083226_21/
drwxr-xr-x 3 ascend ascend 4096 Dec  9 06:05 ge_default_20201209083353_71/
ascend@ubuntu:~/Downloads/dump/npu/20201209083219/0$
```

3. 找到前面记录的名称为name值的文件夹，例如ge_default_20201209083353_71，这些文件即为需要的dump数据文件。



----结束

9.6 Tensor 比对

9.6.1 概述

Tensor比对提供整网比对和单算子比对两种精度比对方式，请根据比对场景选择比对方式。

- 整网比对：将准备好的标准数据文件与待比对数据文件作为输入文件，通过对文件内所有参与计算的算子进行精度比对。
- 单算子比对：在整网比对的基础上指定具体算子名，对单个算子进行详细数据的比对。

标准数据文件与待比对数据文件的准备请参见**9.6.3 整网比对**，具体比对场景请参见**9.6.4 单算子比对**。

9.6.2 约束说明（仅推理场景）

约束

- 需要确保NPU Dump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。
- 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。
- 量化模型里经过量化处理的算子无法比对，必须是反量化的输出才能比对。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”。

说明

- 针对FastRCNN网络场景，ProposalD算子及之后的算子，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。
- 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。
- 精度比对工具推荐环境配置：CPU 8核 2.6Ghz，内存16GB，低于该配置可能会造成比对缓慢。
- 使用量化离线模型在昇腾AI处理器上运行生成的dump数据作为待比对数据时，由于量化与反量化算子之间存在影响精度的中间算子，在比对时会与源数据产生较大的精度损失，所以精度比对时会过滤掉中间算子进行比对，此时输出的结果将不包含整网的所有算子。

9.6.3 整网比对

9.6.3.1 比对操作

步骤1 在MindStudio界面菜单栏选择“Ascend > Model Accuracy Analyzer > New Task”菜单，进入比对界面，如下图所示。

图 9-13 New Task（Linux_Remote Run_1）

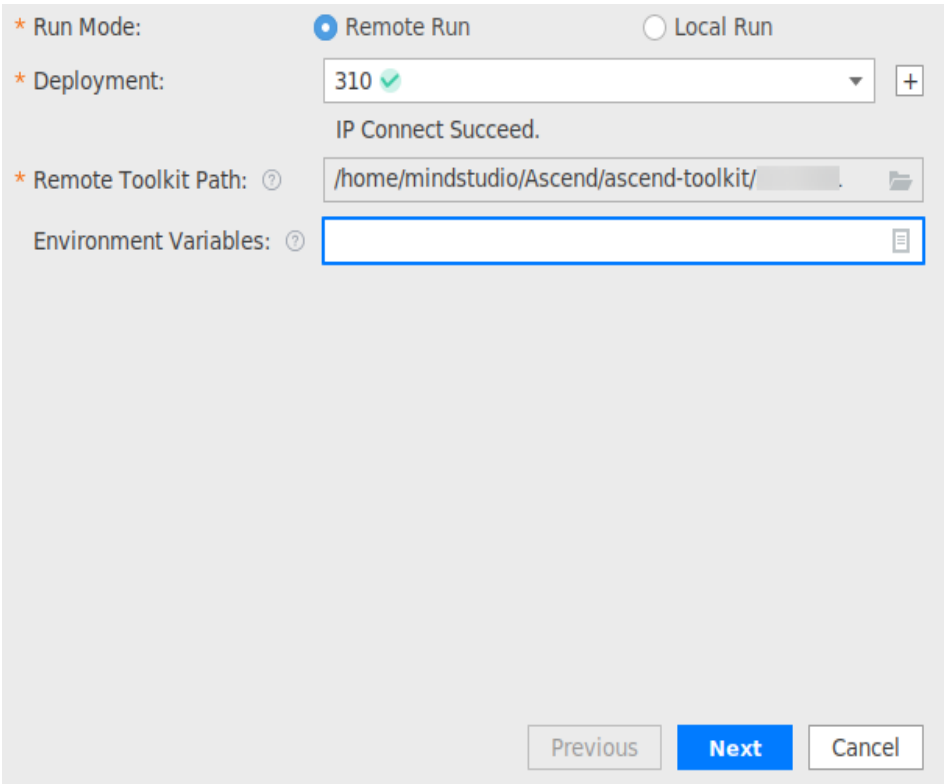


图 9-14 New Task（Linux_Remote Run_2）

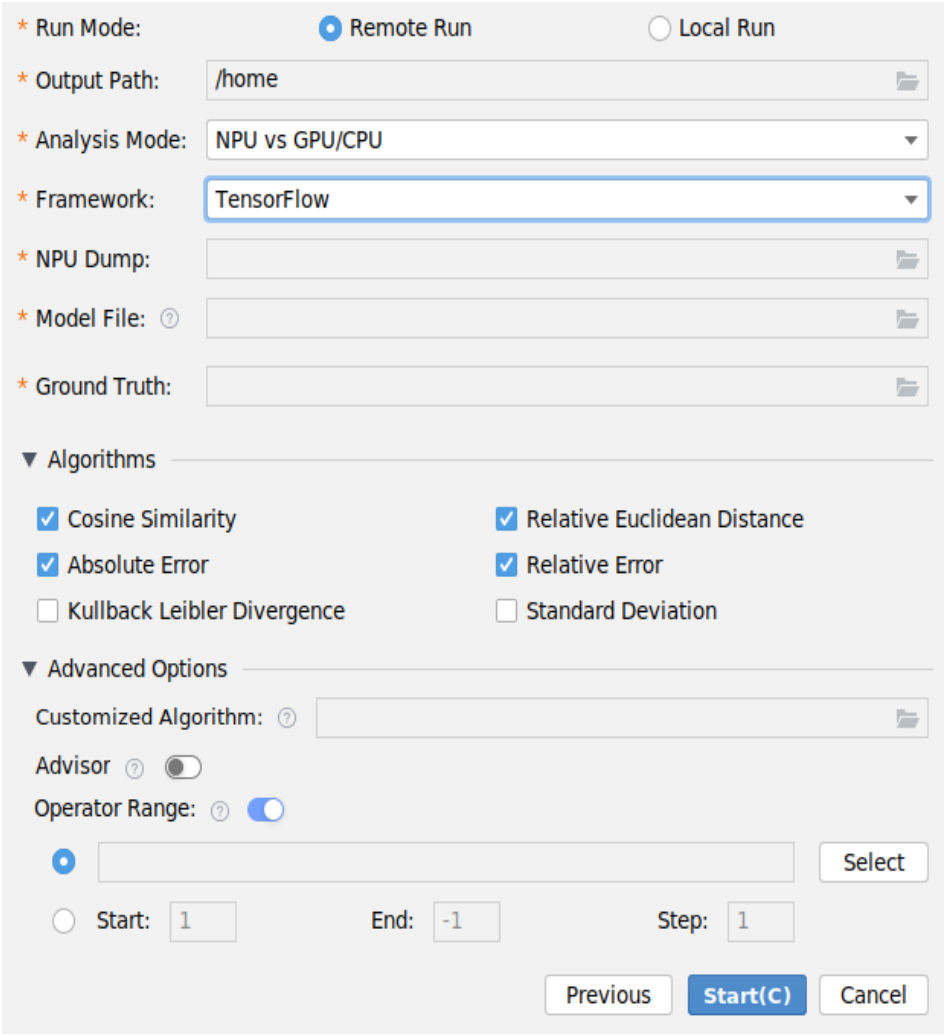


图 9-15 New Task（Linux_Local Run）

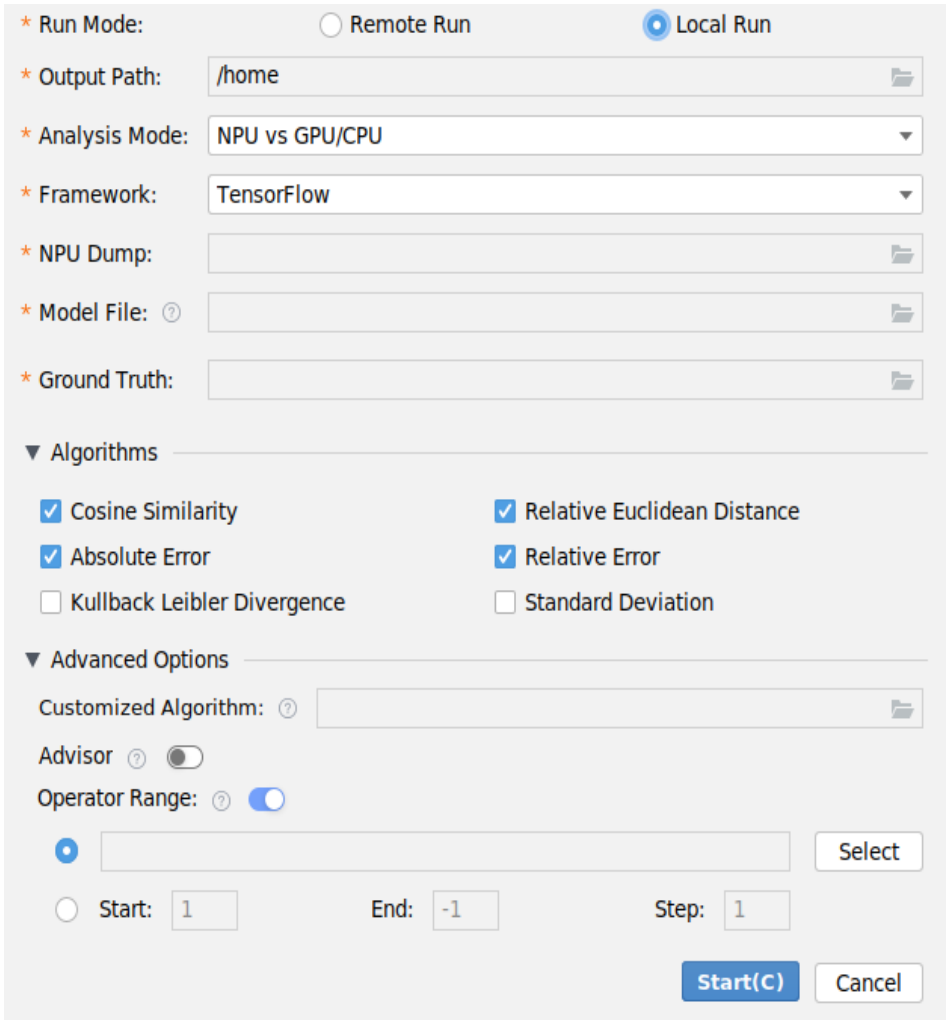


图 9-16 New Task（Windows_1）

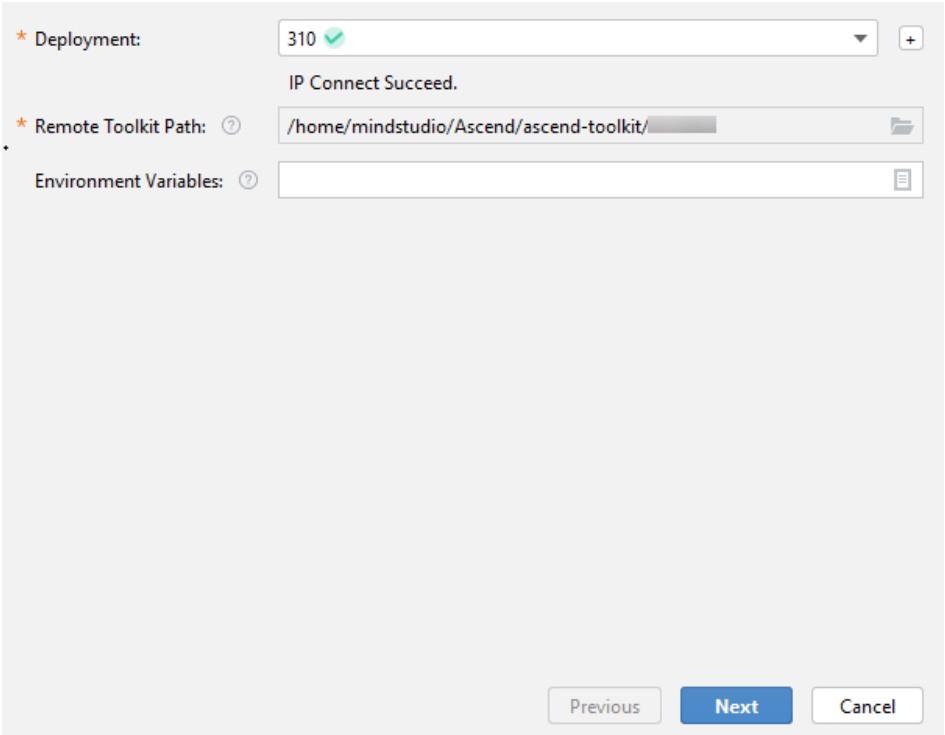
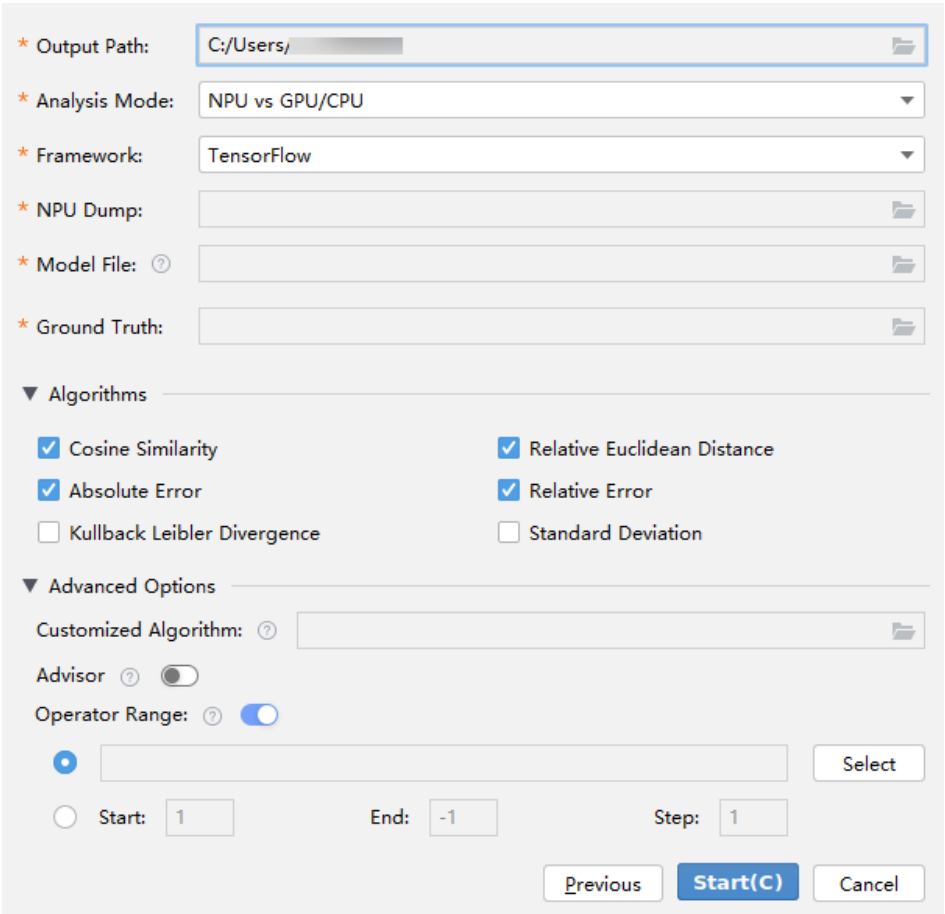


图 9-17 New Task（Windows_2）



步骤2 根据实际需求配置比对参数。各参数详细配置说明请参见**表9-9**。

表 9-9 精度比对 New Task 参数说明

参数	说明
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。 Windows使用场景下仅支持Remote Run，该参数不展示。
Deployment	运行配置，选择Remote Run模式时可见。通过Deployment功能，详细请参见**13.3 Ascend Deployment**，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
Remote 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。例如配置为${HOME}/Ascend/ascend-toolkit/*{version}*/toolkit。 与Deployment参数为绑定关系，单击“Start”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则参数自动填充，可手动修改。
Environment Variables	环境变量配置，选择Remote Run模式时可见，可以直接在框中输入，也可以单击后在弹窗内单击填写。 当Model File指定文件为离线模型文件（*.om）时，需要配置环境变量，否则工具将无法为离线模型文件（*.om）进行ATC转换导致比对失败。 与Deployment参数为绑定关系，单击“Start”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则参数自动填充，可手动修改。
Output Path	比对数据结果存放路径。无论选择Remote Run模式还是Local Run模式，均需要指定为本端路径。默认路径为当前系统的用户目录。
Analysis Mode	精度比对分析模式。可选择模式为： • NPU vs NPU：表示两个比对文件均为昇腾AI处理器上运行生成的dump数据文件。此时Model File参数可选。一般用于分析开启和关闭融合规则时进行模型转换后的dump数据文件之间的精度差。模型转换开启和关闭融合规则的详细介绍请参见**6 模型转换和调优**。 • NPU vs GPU/CPU：表示昇腾AI处理器上运行生成的dump数据文件与原始模型的npy文件进行比对。此时展示Framework必选参数。

参数	说明
Framework	比对数据所属的框架类型。Analysis Mode为NPU vs GPU/CPU时可见。可选类型为： ● TensorFlow：TensorFlow框架模型dump数据的精度比对，支持推理、训练场景，Model File参数必选。 ● ONNX：ONNX框架模型dump数据的精度比对，支持推理场景，Model File参数必选。 ● Caffe：Caffe框架模型dump数据的精度比对，支持推理场景，Model File参数必选。
NPU Dump	昇腾AI处理器上运行生成的dump数据文件目录。 在远端执行比对时（Remote Run），须指定远端设备上的dump数据文件目录。
Model File	模型文件或融合规则文件。 ● Analysis Mode为NPU vs NPU时，进行离线模型转换开启算子融合功能前后的dump数据精度比对，需要指定开融合的算子映射文件（.json）或离线模型文件（*.om）和关融合的算子映射文件（.json）或离线模型文件（*.om）。 ● Analysis Mode为NPU vs GPU/CPU时，根据Framework选择的框架类型选择不同的文件： – TensorFlow：推理场景选择昇腾模型压缩后的量化融合规则文件（json文件）或离线模型文件（*.om）；训练场景选择计算图文件（*.txt）。 – ONNX：选择昇腾模型压缩后的量化融合规则文件（json文件）或离线模型文件（*.om）。 – Caffe：选择昇腾模型压缩后的量化融合规则文件（json文件）或离线模型文件（*.om）。 具体选择文件请参见**9.5.1 比对场景**。
Quantization Rule File（.json）	量化算子映射关系文件（昇腾模型压缩输出的json文件）。仅Framework为Caffe时展示。
Ground Truth	原始模型的npy文件目录。 在远端执行比对时（Remote Run），须指定远端设备上的原始模型的npy文件目录。

参数	说明
Algorithm	比对算法维度。取值为： ● Cosine Similarity：余弦相似度算法，默认勾选。 ● Relative Euclidean Distance：欧氏相对距离算法，默认勾选。 ● Absolute Error，绝对误差，默认勾选，此项执行的比对算法为： – Max Absolute Error：最大绝对误差。 – Mean Absolute Error：平均绝对误差。 – Root Mean Square Error：均方根误差。 ● Relative Error，相对误差，默认勾选，此项执行的比对算法为： – Max Relative Error：最大相对误差。 – Mean Relative Error：平均相对误差。 – Accumulated Relative Error：累积相对误差。 ● Kullback Leibler Divergence：KL散度算法，默认不勾选。 ● Standard Deviation：标准差算法，默认不勾选。 与Customized Algorithm自定义算法之间至少勾选一种算法。
Advance Options	扩展选项。包括Customized Algorithm、Advisor和Operator Range。
Customized Algorithm	自定义算法文件路径。 与Algorithm内置算法之间至少勾选一种算法。 需用户自行准备自定义算法.py文件，所在目录格式为“algorithm”，指定该目录下的自定义算法.py文件，生成自定义算法。自定义算法.py文件相关要求参见《**精度比对工具使用指南**》附录中的“准备自定义算法.py文件”章节。
Advisor	专家系统分析开关，默认关闭。开启后会在完成整网比对后对比对结果进行专家系统分析并输出问题节点、问题类型和优化建议。详细介绍请参见**9.6.3.3 比对结果专家建议**。 本参数要求pandas为1.3或更高版本。 与Operator Range无法同时开启。

参数	说明
Operator Range	设定算子比对范围。有两种设置方式： • 方式一：单击“Select”按钮，在弹出框内勾选需要比对的算子。 • 方式二：根据Start、End、Step参数配置比对算子的范围。 – start：第一个比对的算子，取值范围为[1, 参与计算的算子个数]，默认值为1。 – end：最后一个比对的算子，取值范围为-1或[start, 参与计算的算子个数]，默认值为-1（动态获取网络模型中最后一个参与计算的算子）。 – step：第start+step*n个比对的算子，step取值范围为[1, 参与计算的算子个数)，默认值为1，n为从1开始的正整数。 – 配置格式为：“start,end,step”。比如：-r 1,101,20，表示算子1,21,41,61,81,101的Tensor参与比对。 不配置本参数时，比对网络模型中的所有参与计算的算子。 配置本参数且Analysis Mode参数配置为NPU vs NPU时，需同时指定NPU Dump和Ground Truth的Model File分别指定开融合的算子映射文件（.json）或离线模型文件（*.om）和关融合的算子映射文件（.json）或离线模型文件（*.om）。 与Advisor无法同时开启。

说明

不建议调用与当前用户不一致的其它用户目录下的自定义算法文件（Customized Path）和OM模型文件，避免提权风险。当选择其他用户下的文件时，系统将提示风险。

步骤3 单击“Start”按钮。

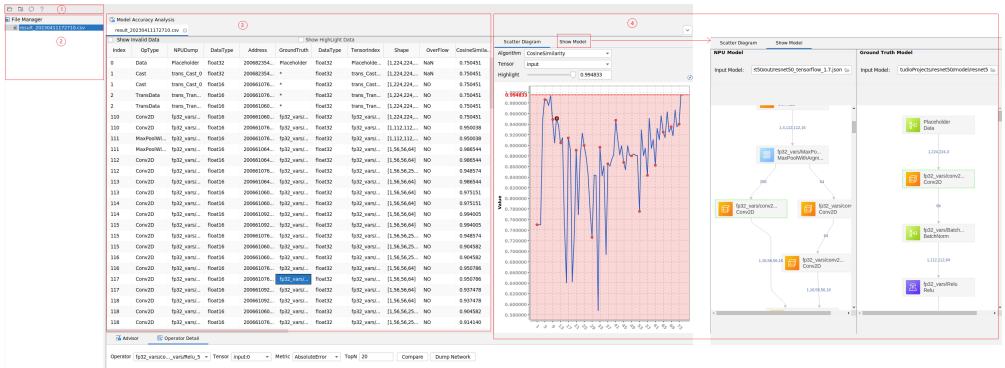
结果说明请参考**9.6.3.2 比对结果**。

----结束

9.6.3.2 比对结果

Tensor比对结果及说明。

图 9-18 Tensor 比对结果



Tensor比对结果界面分为8个区域。其中1~4区域为整网比对结果，如图9-18所示，详细介绍请参见表9-10；5区域为整网比对结果的专家系统分析结果，详细介绍请参见9.6.3.3 比对结果专家建议。6~8区域为单算子比对功能及结果展示，详细介绍请参见9.6.4 单算子比对。

表 9-10 整网比对结果说明

区域	区域名称	说明
1	菜单栏	从左到右分别为Open…、New Task、Refresh、Help四项功能。Open…为打开并展示比对结果csv文件；New Task为创建新的比对任务；Refresh用于读取并刷新File Manager中管理的文件；单击Help弹出小窗，可展示精度比对工具的使用限制（Restrictions）、使用建议、在线教程链接等。
2	File Manager，历史数据管理	显示用户指定文件夹以及文件夹下生成的整网比对的csv文件以及显示通过Open…单独打开的csv文件；对文件夹和csv，提供历史数据管理功能，包括打开、删除、另存为；在文件夹处右键删除；在空白处右键创建新比对任务（New Task）、刷新（Refresh）和Open…（打开并展示比对结果csv文件）。
3	Model Accuracy Analysis，精度比对分析界面	默认仅显示有结果的算子。可单击列名，进行排序，各列字段含义请参见表9-11。 单击勾选Show Invalid Data，可展示无法比对的数据；单击勾选Show Highlight Data，仅展示不符合Highlight阈值的比对结果数据；去勾选Show Highlight Data状态下，滑动右侧Highlight游标，则表格中不符合Highlight阈值的数据标红；Show Highlight Data由勾选转变为去勾选的状态，表格还原到默认状态。
4	Scatter Diagram，各项算法指标的散点分布图 Show Model，比对模型可视化展示	Scatter Diagram：横坐标表示算子的执行顺序，纵坐标为算法指标在对应Tensor上的实际取值。各字段含义请参见表9-12。 Show Model：分别展示NPU和Ground Truth的模型图。详细介绍请参见表9-13。
注：历史数据管理功能暂不支持数据老化，即当磁盘空间不足时系统无法自动删除多余的历史文件，请用户自行删除多余的文件以保证比对程序的运行。		

表 9-11 比对结果字段说明

字段	说明
Index	网络模型中算子的ID。
OpSequence	算子运行的序列。全网层信息文件中算子的ID。仅配置“Operator Range”时展示。

字段	说明
OpType	算子类型。
NPUDump	表示NPU Dump模型的算子名。光标悬浮时，可显示具体算子所在的文件路径。
DataType	表示NPU Dump侧数据算子的数据类型。
Address	dump tensor的虚拟内存地址。用于判断算子的内存问题。仅基于昇腾AI处理器运行生成的dump数据文件在整网比对时可提取该数据。
GroundTruth	表示Ground Truth模型的算子名。光标悬浮时，可显示具体算子所在的文件路径。
DataType	表示Ground Truth侧数据算子的数据类型。
TensorIndex	表示NPU Dump模型算子的input ID和output ID。
Shape	比对的Tensor的Shape。
OverFlow	溢出算子。显示YES表示该算子存在溢出；显示NO表示算子无溢出；显示NaN表示不做溢出检测。 开启Advisor功能时展示，为**9.6.3.3 比对结果专家建议**的**FP16溢出检测**专家建议提供数据。
CosineSimilarity	进行余弦相似度算法比对出来的结果。取值范围为[-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。
RelativeEuclideanDistance	进行欧氏相对距离算法比对出来的结果。取值范围为0到无穷大，值越接近于0，表明越相近，值越大，表明差距越大。
MaxAbsoluteError	进行最大绝对误差算法比对出来的结果。取值范围为0到无穷大，值越接近于0，表明越相近，值越大，表明差距越大。
MeanAbsoluteError	表示平均绝对误差（MAE），取值范围为0到无穷大。 • MeanAbsoluteError趋于0，RootMeanSquareError趋于0，说明测量值与真实值越近似。 • MeanAbsoluteError趋于0，RootMeanSquareError越大，说明存在局部过大的异常值。 • MeanAbsoluteError越大，RootMeanSquareError等于或近似MeanAbsoluteError，说明整体偏差越集中。 • MeanAbsoluteError越大，RootMeanSquareError越大于MeanAbsoluteError，说明存在整体偏差，且整体偏差分布分散。 • 不存在以上情况的例外情况，因为RootMeanSquareError ≥ MeanAbsoluteError恒成立。

字段	说明
RootMeanSquareError	表示均方根误差（RMSE），取值范围为0到无穷大。 ● MeanAbsoluteError趋于0，RootMeanSquareError趋于0，说明测量值与真实值越近似。 ● MeanAbsoluteError趋于0，RootMeanSquareError越大，说明存在局部过大的异常值。 ● MeanAbsoluteError越大，RootMeanSquareError等于或近似MeanAbsoluteError，说明整体偏差越集中。 ● MeanAbsoluteError越大，RootMeanSquareError越大于MeanAbsoluteError，说明存在整体偏差，且整体偏差分布分散。 ● 不存在以上情况的例外情况，因为RootMeanSquareError ≥ MeanAbsoluteError恒成立。
MaxRelativeError	表示最大相对误差。取值范围为0到无穷大，值越接近于0，表明越相近，值越大，表明差距越大。
MeanRelativeError	表示平均相对误差。取值范围为0到无穷大，值越接近于0，表明越相近，值越大，表明差距越大。
AccumulatedRelativeError	进行累积相对误差算法比对出来的结果。取值范围为0到无穷大，值越接近于0，表明越相近，值越大，表明差距越大。
StandardDeviation	进行标准差算法比对出来的结果。取值范围为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NPU Dump和Ground Truth两组数据的均值和标准差，第一组展示NPU Dump模型dump数据的数值(均值;标准差)，第二组展示Ground Truth模型dump数据的数值（均值;标准差）。
KullbackLeiblerDivergence	进行KL散度算法比对出来的结果。取值范围为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。
CompareFailReason	算子无法比对的原因。 若余弦相似度为1，则查看该算子的输入或输出shape是否为空或全部为1，若为空或全部为1则算子的输入或输出为标量，提示：this tensor is scalar。
注1：余弦相似度和KL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；KL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。 注2：光标悬浮在表头可以看到对应的参数详细解释。 注3：若配置了自定义算法比对，则在比对结果的内置算法后增加对应自定义算法列。 注4：单击表中任意结果单元格，右侧散点分布图或模型图可跳转到对应算子并高亮；单击右侧Show Model的模型中任意算子，左侧表中对应算子的单元格高亮。	

表 9-12 散点分布图字段说明

字段	说明
Algorithm	选择展示对应比对算法结果的散点分布图，不支持展示StandardDeviation、KullbackLeiblerDivergence和AccumulatedRelativeError。
Tensor	过滤显示Input、Output结果散点分布图。
Highlight	算子精度阈值。通过滑动游标在对应算法指标的[min,max]间滑动来设置算法指标（纵坐标）的阈值，符合阈值的点显示为蓝色，不符合阈值的点显示为红色，同时左侧表格不符合阈值的数据标红。对于CosineSimilarity算法来说低于阈值的数据为不符合；对于其他算法来说，高于阈值的数据为不符合。
注1：光标移动到对应Tensor点上时，浮窗显示Tensor信息。信息包括：Index（Tensor对应算子的Index）、Op Name（算子名称）、Tensor Index（Tensor类型（input/output））以及Value（在当前算法维度下的Tensor数值）。 注2：支持对散点图进行缩放。 注3：指定区域3中的Tensor时，高亮对应Tensor点。	

表 9-13 比对模型可视化展示字段说明

字段	说明
NPU Model	离线模型可视化。指定算子映射文件（.json）或离线模型文件（*.om）展示。 训练场景下，若整网比对使用的Model File为计算图文件（*.txt），此处不支持展示模型图。
Ground Truth Model	原始模型可视化。指定原始模型文件展示。
Input Model	指定算子映射文件（.json）、离线模型文件（*.om）或原始模型文件。
注1：选中区域3中的某个Tensor时，模型网络跳转到对应节点并高亮。 注2：滑动散点分布图中的Highlight游标，模型网络中不符合Highlight阈值的节点标红。	

9.6.3.3 比对结果专家建议

9.6.3.3.1 概述

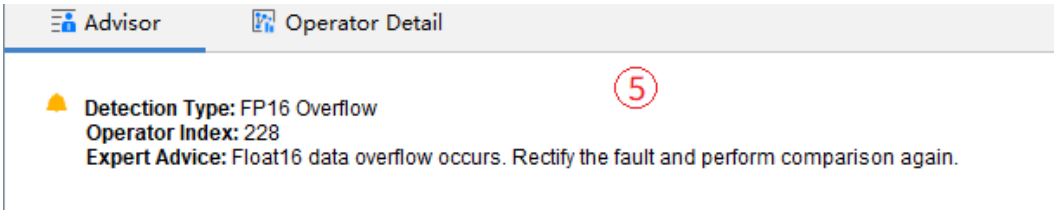
精度比对工具本身只提供自有实现算子在昇腾AI处理器上的运算结果与业界标准算子的运算结果的差异比对功能，而输出的比对结果需要用户自行分析并找出问题。而对结果的分析工作对于用户来说也是一大难点，本节提供专家系统工具为用户提供精度比对结果的结果分析功能，有效减少用户排查问题的时间。只需在**9.6.3.1 比对操作**配置任务时勾选“Advisor”选项，系统则会在比对完成后自动进行结果文件的分析，并输出优化建议。

当前支持的分析检测类型有：FP16溢出检测、输入不一致检测、整网一致性检测（整网一致性检测包括：问题节点检测、单点误差检测和一致性检测三个小点）

9.6.3.3.2 分析结果说明

专家系统分析结果展示在比对结果的**5区域**。

图 9-19 专家系统分析结果（仅为示例）



Float16 溢出检测

针对比对数据中数据类型为Float16的数据，进行溢出检测。如果存在溢出数据，输出专家建议。

专家系统分析结果:

Detection Type: FP16 overflow

Operator Index: 228

Expert Advice: Float16 data overflow occurs. Rectify the fault and perform comparison again.

检测类型：Float16溢出检测

Operator Index：228

专家建议：存在Float16数据溢出，请修正溢出问题，再进行比对。

输入不一致检测

针对整网的输入数据进行检测，主要判断整网两批待比对数据的输入data是否一致。如果存在不一致问题（余弦相似度<0.99），输出专家建议。

专家系统分析结果:

Detection Type: Input inconsistent

Operator Index: 0

Expert Advice: The input data of NPUDump is inconsistent with that of GroundTruth. Use the same data or check the data preprocessing process.

检测类型：输入不一致检测

Operator Index：0

专家建议：NPUDump和GroundTruth间的输入数据不一致，请使用相同数据或者检查数据预处理流程。

整网一致性检测（问题节点检测）

判断整网比对结果中，是否某层小于阈值，该层后续数据均小于阈值或最后一层小于阈值（余弦相似度<0.99），输出量化误差修正建议。

专家系统分析结果：

Detection Type: global consistency

Operator Index: 1174

Expert Advice: The accuracy of some tensors is low, resulting in an unqualified final accuracy. This may be caused by quantization. Calibrate the data or contact Huawei for further diagnosis.

检测类型：整网一致性检测

Operator Index：1174

专家建议：部分张量精度较低，且导致最终结果精度不达标；很可能由量化造成，请进行数据校准或者反馈给华为做进一步定位。

整网一致性检测（单点误差检测）

判断整网比对结果中，是否某层小于阈值（余弦相似度<0.99），但最终结果符合精度要求，输出专家建议。

专家系统分析结果：

Detection Type: global consistency

Operator Index: 195

Expert Advice: The accuracy of some tensors is low, while the final accuracy is qualified. This may be caused by Ascend internal optimization. Ignore or contact Huawei for further diagnosis.

检测类型：整网一致性检测

Operator Index：195

专家建议：部分张量精度较低，但最终结果精度达标，可能由内部优化导致，请忽略或反馈给华为做进一步定位。

整网一致性检测（一致性检测）

比对结果中的所有数据均符合精度要求，输出专家建议。

专家系统分析结果：

Detection Type: global consistency

Operator Index: NA

Expert Advice: All data in the comparison result meets the accuracy requirements.

If data accuracy of the model is still not up to standard in practical application, please check the post-processing process of model outputs.

检测类型：整网一致性检测

Operator Index：NA

专家建议：比对结果中的所有数据均符合精度要求。

如果模型实际应用中，精度依旧不达标，请排查输出数据的后处理流程。

9.6.4 单算子比对

单算子比对界面说明

单算子比对功能是在**9.6.3.2 比对结果**的基础上选择具体算子并根据需要指定参数进行比对的。界面展示为Tensor比对结果界面的**6区域**、**7区域**和**8区域**，如**图9-20**所示。详细介绍请参见**表9-14**。

图 9-20 单算子比对

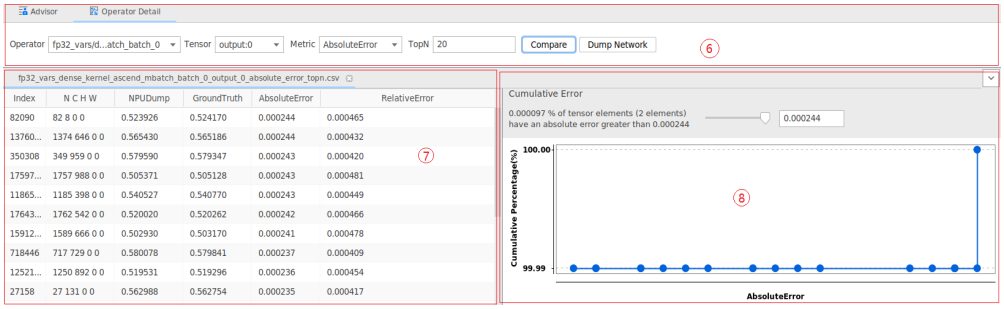


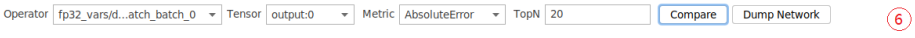
表 9-14 单算子比对结果说明

区域	说明
6	Operator Detail，单算子比对功能。下发单算子比对命令。具体操作请参见**比对操作**。 其中Dump Network按钮为子模型导出，单击后在MindStudio主界面弹出Dump Network界面，子模型导出详细介绍请参见**9.6.6.3 子模型导出与比对**。
7	单算子比对TopN结果。各列字段解释请参见**表9-15**。
8	Cumulative Error，对于TopN Tensor元素，绘制的误差累积分布折线图。详细介绍请参见**表9-15**。

比对操作

单算子比对操作功能界面展示为**图9-18**的**6区域**，如**图9-21**所示。操作步骤如下：

图 9-21 单算子比对操作



步骤1 基于已经读取到的Tensor信息，通过Operator下拉框选择要比对的算子和对应Tensor（output/input）。

步骤2 通过Metric选择比对算子的Absolute Error（绝对误差）或Relative Error（相对误差）。

步骤3 配置输出该指标的TopN个Tensor元素（取值范围为[1,10000]）。

步骤4 单击“Compare”按钮进行单算子比对。

TopN结果显示请参见**比对结果**。

----结束

比对结果

单算子比对结果界面展示为**图9-18**的**7区域**和**8区域**，如**图9-22**所示。详细介绍请参见**表9-15**。

图 9-22 单算子比对结果

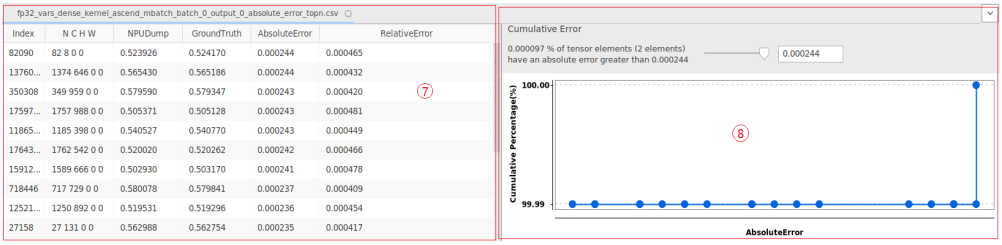


表 9-15 比对结果字段说明

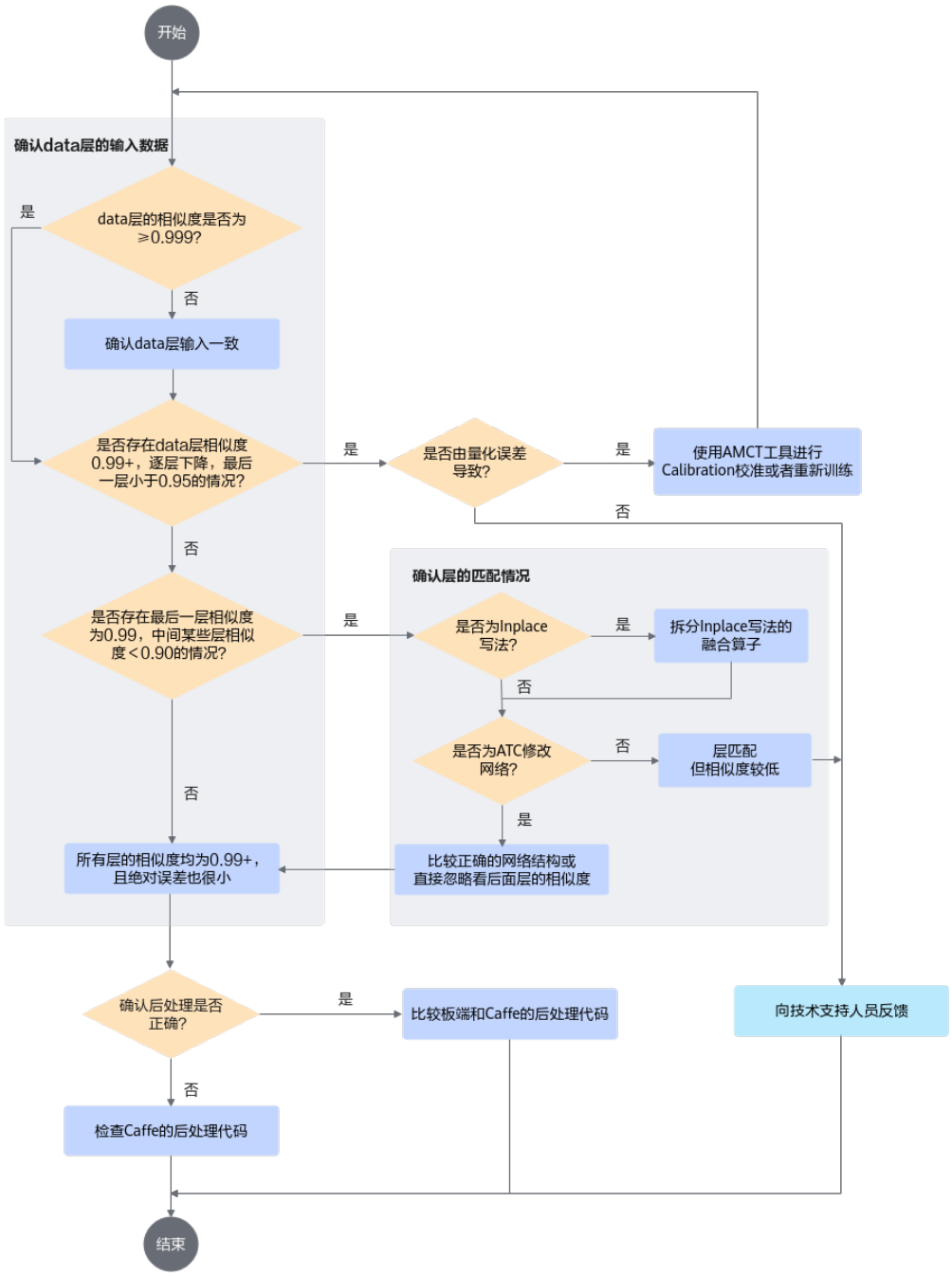
字段	说明
7区域	
Index	算子比对的条数。
N C H W	数据格式。
NPUDump	NPU Dump侧算子的dump值。
GroundTruth	Ground Truth侧算子的dump值。
AbsoluteError	绝对误差，NPU Dump侧算子的dump值减Ground Truth侧算子的dump值取绝对值比对出来的结果。小数点后最多6位。
RelativeError	相对误差，Absolute Error值除以Ground Truth侧算子的dump值比对出来的结果。当Ground Truth侧算子的dump值为0时，该处显示为“-”。小数点后最多6位。
8区域	
A % of tensor elements (B elements) have an absolute error greater than C.	当前算子比对的所有tensor元素的绝对误差结果中有A %的tensor也就是B个元素的绝对误差超过了C。其中absolute error根据**5区域**配置的Metric取值变化可以为relative error；右侧滑块控制C的取值，范围由**6区域**AbsoluteError或RelativeError的最大最小值决定。

字段	说明
误差累积分布折线图	横坐标为**6区域**的AbsoluteError或RelativeError，取值范围由AbsoluteError或RelativeError的最大最小值决定；纵坐标为累积百分占比，含义为AbsoluteError或RelativeError到达某个阈值时，小于等于该阈值的所有tensor元素在整体tensor元素中的占比。
注：光标移动到对应Tensor点上时，浮窗显示Tensor信息。信息包括：Index（算子比对的条数）、Absolute Error/Relative Error（绝对/相对误差）、Cumulative Percentage（累积百分占比）。	

9.6.5 精度问题分析流程

精度比对完成后若存在精度问题，则需要进行问题分析并找出解决方案，图9-23以Caffe推理应用场景为例提供精度问题分析基本思路参考。

图 9-23 精度问题分析基本思路



以下介绍图9-23中各步骤的具体操作：

1. 确认第一层data层的输入数据是否正确：找一张有精度问题的图片，编译、指令仿真、Caffe都用这张图片。比较仿真和Caffe每层的相似度。而后按照上图中顺序进行判断。
2. 确认data层的相似度是否为≥0.999：
   - 是：执行下一个判断。
   - 否：确认data层输入一致。检查均值[mean_file]、缩放[data_scale]、预处理方式[norm_type]，是否和Caffe一致。（Mxnet和Darknet(yolo)）网络训练时，默认是RGB，所以转换模型时需要配置[RGB——order]为RGB。

3. 确认是否存在data层相似度0.99+，逐层下降，最后一层小于0.95的情况：
   - 是：确认是否量化误差导致。

     修改ATC的配置项，配置[dump_data]为1，输出校准数据到mapper_quant目录。

     配置[forward_quantization_option]为1，即只做数据量化，权重不量化。比较mapper_quant和caffe的相似度，如果相似度达标，说明是权重量化误差导致。

     配置[forward_quantization_option]为2，即只做权重量化，数据不量化。比较mapper_quant和caffe的相似度，如果相似度达标，说明是数据量化误差导致。

     - 是：使用AMCT工具进行Calibration校准或者重训。
     - 否：向技术支持人员反馈问题。
   - 否：执行下一个判断。
4. 确认是否存在最后一层相似度为0.99，中间某些层相似度＜0.90的情况：
   - 是：确认层的匹配情况。

     ATC优化网络结构，以适应硬件执行，所以相似度比较时有可能层和Caffe不匹配。

     i. 判断是否为Inplace写法，即top与bottom名称相同。某些层不支持Inplace，需要拆分。例如conv + tanh，在Caffe支持Inplace，只输出tanh的数据，而ATC不支持conv与tanh融合，分别输出conv和tanh，所以比较时需要注意。

     ii. 判断是否为ATC修改网络。请查看cnn_net_tree.dot（ATC编译时生成），和原来的prototxt比较，是否修改了网络结构。如SPP层拆分为Pooling和Concat，所以要与Concat的结果比，或直接观察后面层的相似度（即执行下一个判断）。

     iii. 层匹配，但相似度较低，向技术支持人员反馈问题。
   - 否：执行下一个判断。
5. 如果所有层的相似度均为0.99+且绝对误差也很小，则可能是后处理的问题，确认后处理是否正确。

   假设Caffe的结果经过Caffe的后处理画框或分类，则把仿真结果也使用Caffe的后处理，观察是否正常画框或分类。

   如果正常，则说明是板端后处理问题，请比较板端和Caffe的后处理代码。

   如果不正常，而数据的相似度为0.99且绝对误差很小，则说明Caffe的后处理代码对数据很敏感，请检查Caffe的后处理代码。
6. 向技术支持人员反馈问题。

   请将以下信息反馈给技术支持人员：
   - 如果是Caffe则返回prototxt、caffemodel，如果是PyTorch则返回ONNX模型和py定义，pth文件。如果不方便提供模型，请提供问题层对应的prototxt、py、pth、weights和输入/输出数据。
   - 编译用的参数、图片、均值文件。
   - 编译时打印的ATC版本号，如：Mapper Version 1.0.0.0_B010(PICO_1.0) 2110161033840ed952(CPU)(INST_2.0.9)

9.6.6 扩展功能

9.6.6.1 dump 数据文件转换为 npy 数据文件

MindStudio精度比对工具提供dump数据文件转换为npy数据文件功能，用于用户根据自身需求将昇腾AI处理器生成的dump数据文件转换成npy数据文件，方便查看或进行9.6.6.2 npy与npy文件间的精度比对。

步骤1 在MindStudio界面菜单栏选择“Ascend > Model Accuracy Analyzer > Covert”菜单，进入转换配置界面，如下图所示，界面中参数解释如表9-16所示。

图 9-24 转换配置（Local Run）

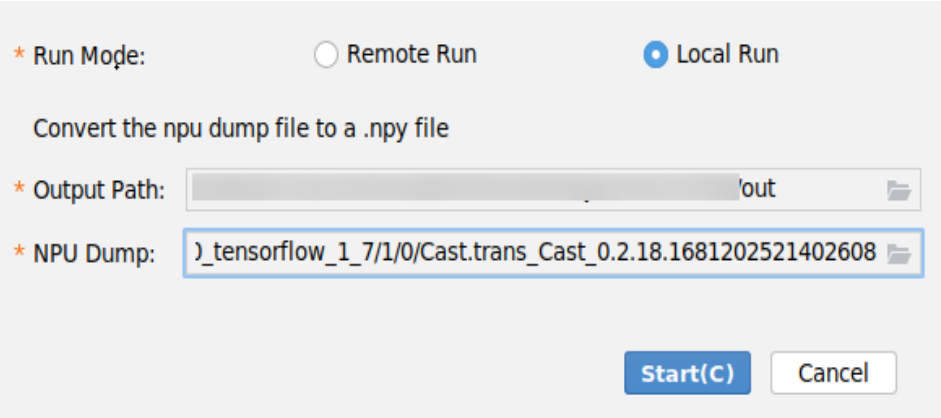
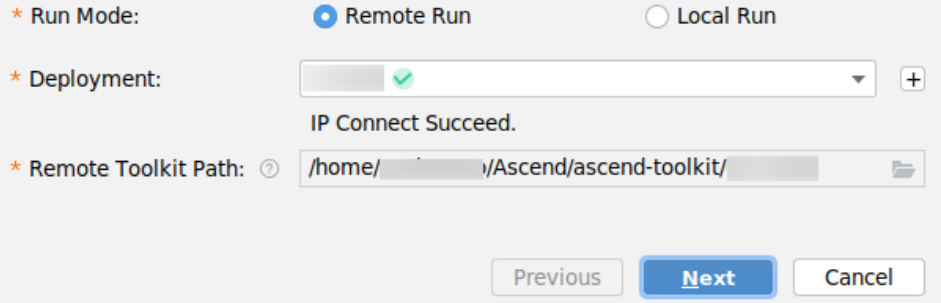


图 9-25 运行配置（Remote Run）



Remote Run模式下，配置完运行配置后，单击“Next”进入到转换配置界面，如图9-26所示。

图 9-26 转换配置（Remote Run）

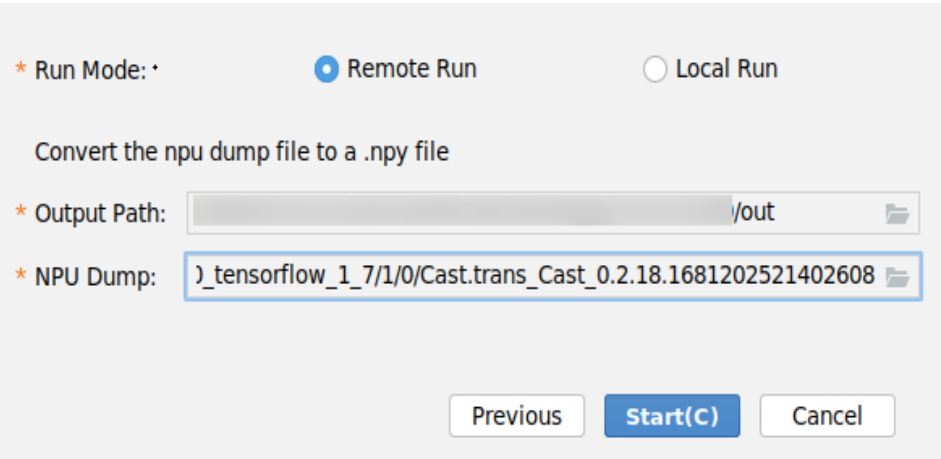
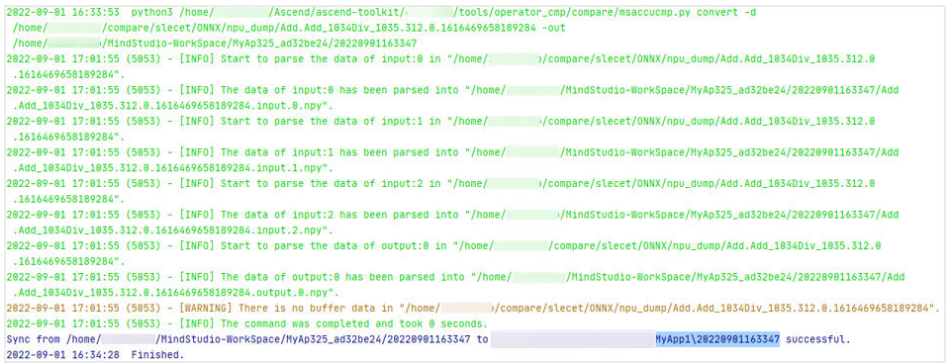


表 9-16 参数说明

参数	说明
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。 Windows使用场景下仅支持Remote Run，该参数不展示。
Deployment	运行配置，选择Remote Run模式时可见，必选配置。通过Deployment功能，详细请参见**13.3 Ascend Deployment**，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
Remote Toolkit Path	远端运行环境Ascend-cann-toolkit软件包安装路径，选择Remote Run模式时可见，必选配置。例如配置为${HOME}/Ascend/ascend-toolkit/*{version}*/toolkit。 与Deployment参数为绑定关系，单击“Next”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则参数自动填充，可手动修改。
Output Path	转换成功后.npy文件输出本地路径，用户自行配置。
NPU Dump	待转换的dump文件。 当前仅支持一次转换一个dump文件。

步骤2 单击“Start”，在MindStudio界面下方Output栏显示转换完成信息，如**图9-27**所示。

图 9-27 Output



比对结果存放在**表9-16**中“Output Path”配置的路径。

----结束

9.6.6.2 npy 与 npy 文件间的精度比对

概述

MindStudio精度比对工具支持单个npy与npy文件之间的精度比对。基本要求如下：

- 对于dump数据文件需要先完成**9.6.6.1 dump数据文件转换为npy数据文件**。
- 需要确保两个比对文件内的Shape一致。
- 仅支持CosineSimilarity（余弦相似度）、MaxAbsoluteError（最大绝对误差）、AccumulatedRelativeError（累积相对误差）、RelativeEuclideanDistance（欧氏相对距离）、MeanAbsoluteError（平均绝对误差）、RootMeanSquareError（均方根误差）、MaxRelativeError（最大相对误差）、MeanRelativeError（平均相对误差）比对算法。

操作步骤

步骤1 在MindStudio界面菜单栏选择“Ascend > Model Accuracy Analyzer > File Comparison”菜单，进入比对参数配置界面，如下图所示，比对界面参数说明如**表9-17**所示。

图 9-28 npy 文件比对配置（Local Run）

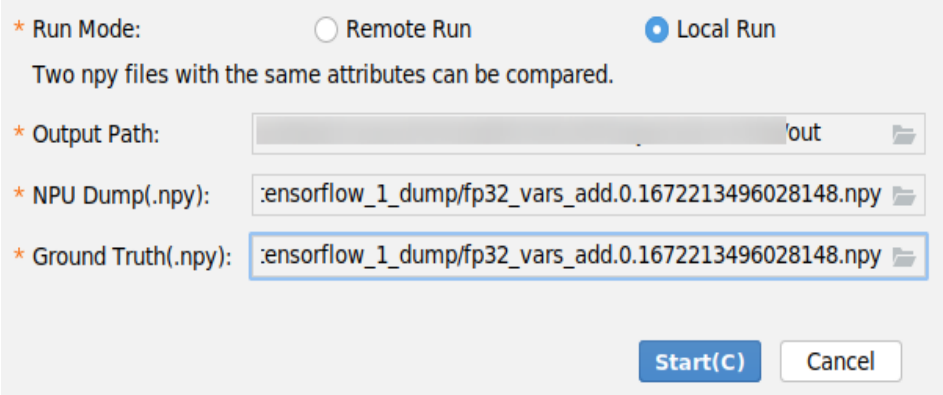
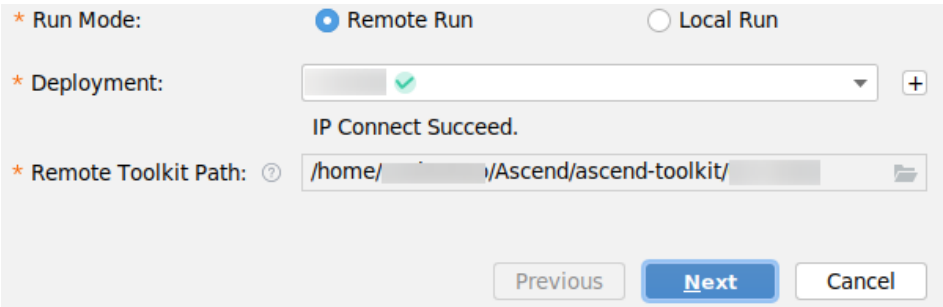


图 9-29 运行配置（Remote Run）



Remote Run模式下，配置完运行配置后，单击“Next”进入到比对配置界面，如**图9-30**所示。

图 9-30 npy 文件比对配置（Remote Run）

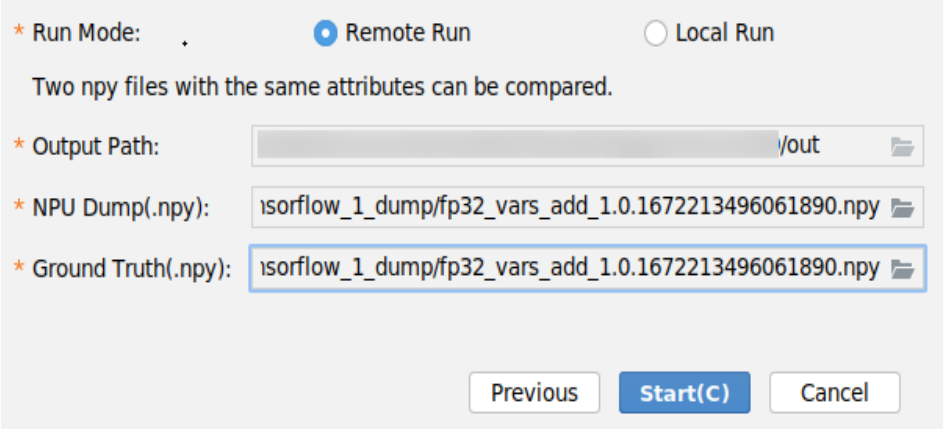


表 9-17 参数说明

参数	说明
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。 Windows使用场景下仅支持Remote Run，该参数不展示。
Deployment	运行配置，选择Remote Run模式时可见。通过Deployment功能，详细请参见**13.3 Ascend Deployment**，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
Remote Toolkit Path	远端运行环境Ascend-cann-toolkit软件包安装路径，选择Remote Run模式时可见。例如配置为${HOME}/Ascend/ascend-toolkit/*{version}*/toolkit。 与Deployment参数为绑定关系，单击“Next”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则参数自动填充，可手动修改。
Output Path	比对结果输出本地路径，用户自行配置。
NPU Dump(.npy)	待比对的npy文件。 在远端执行比对时（Remote Run），须指定远端设备上的npy数据文件。
Ground Truth(.npy)	待比对的npy标杆数据文件。

步骤2 单击“Start”，在MindStudio界面下方Output栏显示比对完成信息，如**图9-31**所示。

图 9-31 Output

比对结果存放在**表9-17**中“Output Path”配置的路径。

----结束

9.6.6.3 子模型导出与比对

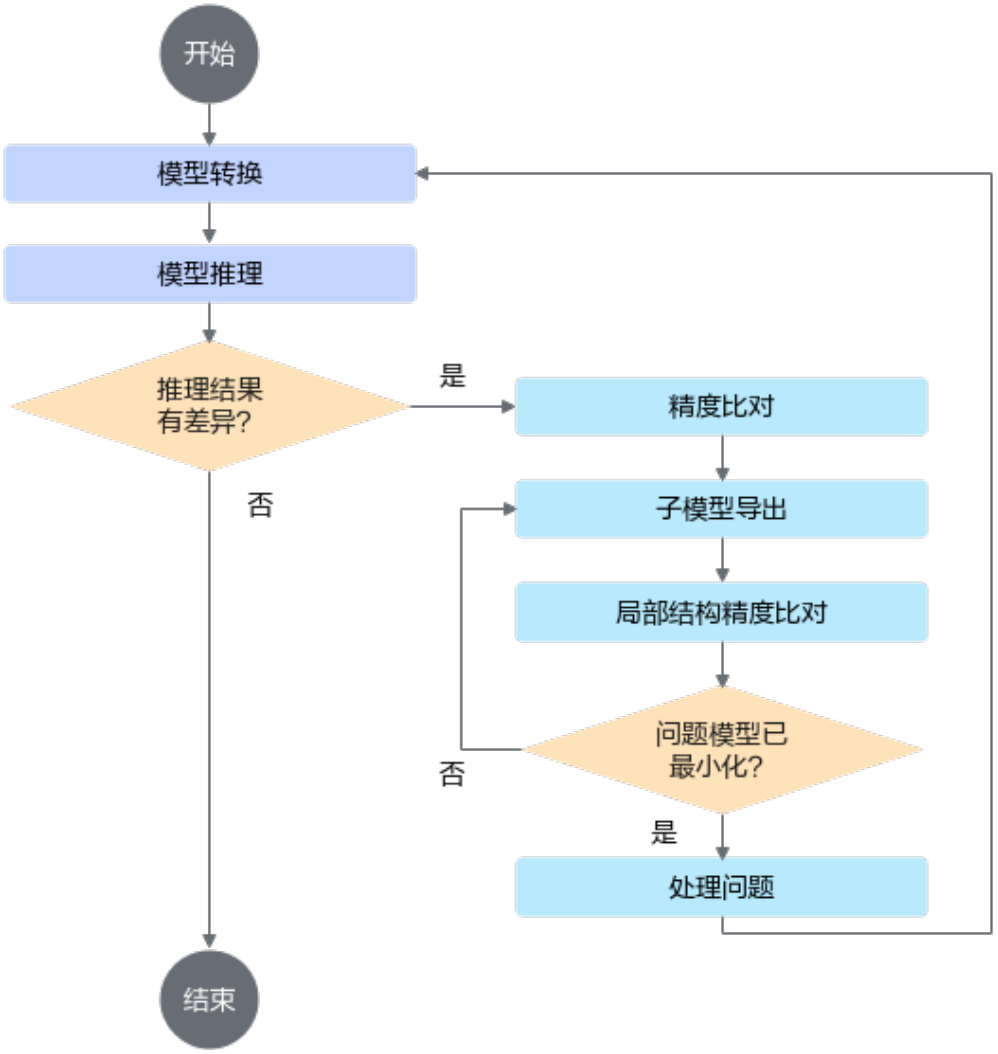
概述

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，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9-32**所示。

说明

- 若需使用TensorFlow框架的子模型导出功能，请自行安装TensorFlow 1.15 AI框架和Python3（版本要求：3.7.5~3.7.11）。
- 若需使用Caffe框架的子模型导出功能，请自行安装Caffe AI框架并确保框架能在Python3.7或Python3.9下正常运行，其中yolov2、yolov3以及量化后的网络模型暂不支持在Caffe AI框架下的子网导出，Caffe AI框架支持的操作系统请参见**Caffe官网**中“Installation”模块；Windows系统不支持使用Caffe框架的子模型导出功能。

图 9-32 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**6 模型转换和调优**。
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. 针对最小模型进行优化。

   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

在MindStudio界面，依次单击菜单栏“Ascend > Dump Network”，进入子模型导出界面。

导出步骤

步骤1 在MindStudio界面菜单栏选择“Ascend > Dump Network”进入子模型导出界面，如**图9-33**所示。

图 9-33 子模型导出界面

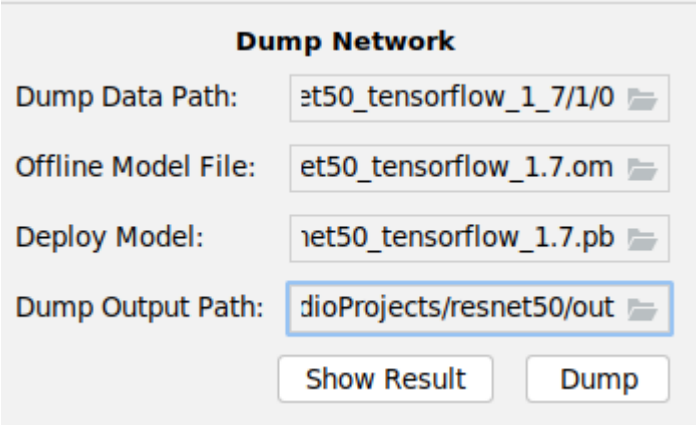


表 9-18 界面参数以及按钮说明

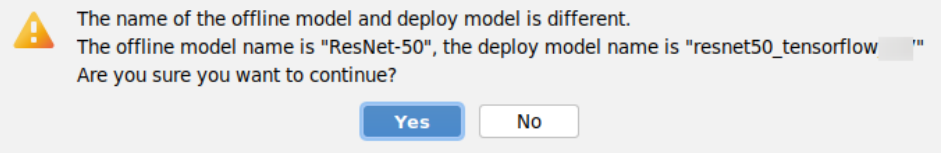
参数	说明
Dump Data Path	精度比对界面“NPU Dump”中配置的离线模型对应的dump数据。单击右侧的选择dump目录。
Offline Model File	精度比对界面“Model File”中配置的离线模型文件。单击右侧的选择离线模型文件。
Deploy Model	• 原始网络模型文件，单击右侧的选择MindStudio安装服务器相应文件。 • 如果为昇腾模型压缩工具量化后的模型文件，则此处需要选择量化后可在昇腾AI处理器部署的模型文件。
Deploy Weight	• 原始网络模型权重文件，若模型文件与权重文件在MindStudio安装服务器相同位置，选择模型文件后，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2 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9-34所示界面。

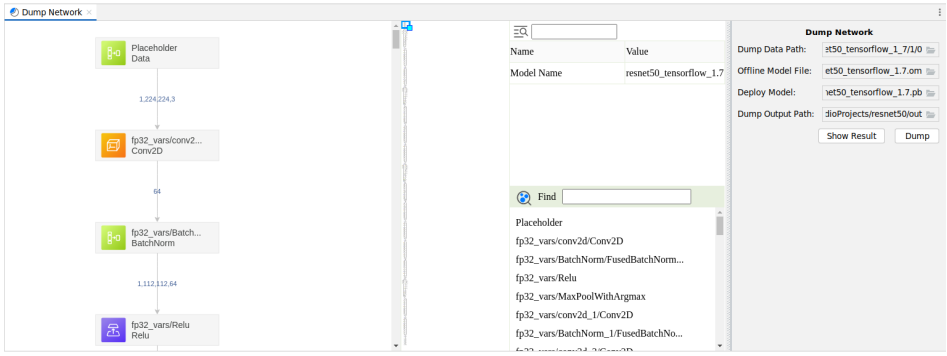
- 单击“Yes”继续解析，进入图9-35。
- 单击“No”，返回子模型导出界面。

图 9-34 Model Name 取值不同



解析成功的原始网络模型如图9-35所示。

图 9-35 解析成功结果



其中：

- **查看算子信息**

  单击图9-35中间区域框下方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。

- **查看算子输出shape信息**

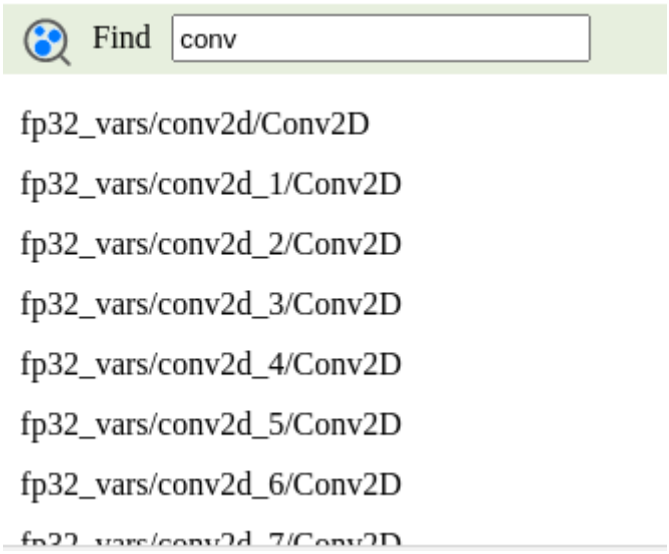
  图9-35还展示了每一层算子输出的维度和shape信息，如每一层算子连接线中间的NCHW，1,3,224,224等信息。

- **搜索算子**

  在图9-35所示界面，中间下方为搜索区域。

  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。

图 9-36 搜索算子内部信息



- **搜索算子内部信息**

  在图9-35中的输入框输入想要查询的信息，之后单击即可逐个查找。

步骤3 选中怀疑有问题的局部网络模型第一个节点（输入节点），右键选择“start”；选中最后一个节点（输出节点）右击选择“end”，“start”与“end”中间的节点则为蓝色选中状态。

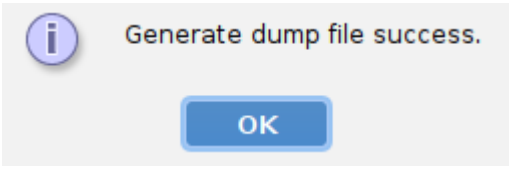
- 选中了输入和输出节点后，如果想更换输入或输出节点，则在新的节点上右击选择相应的属性即可，新的节点颜色随之变化，原有节点恢复原始颜色。
- 如果在某个节点选择“start”，并在该节点上方节点选择“end”，则无法选择。

步骤4 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9-37所示信息，则说明导出成功。

图 9-37 导出成功提示



----结束

查看导出成功后结果

步骤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。

步骤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/username/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3 用户可以将导出的子模型再次进行精度比对，如果问题解决则进入下一步**导出子模型重新执行推理**验证。

----结束

导出子模型重新执行推理

步骤1 将**步骤1**导出的子模型，参见**6 模型转换和调优**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。

步骤2 将**步骤1**导出的子模型“start”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。

1. 查找“start”层的输入数据。

   输入数据为子模型导出时“Dump Layer”为“start”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”层的输入数据。

2. 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/latest/**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 shape_conversion.py -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。

3. 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/username/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/username/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 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。

步骤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1**和**步骤2**中的离线模型和输入数据在昇腾AI处理器执行推理业务，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1**中导出的原始网络子模型与**步骤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。

步骤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，针对最小模型进行优化。

----结束

9.7 附录

9.7.1 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，本文提供以下脚本将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/tools/operator_cmp/compare目录。

步骤3 执行msaccucmp.py脚本，转换dump文件为numpy文件。举例：

```
python3 msaccucmp.py convert -d dump_file [-out output] [-f format -s shape] [-o output_tensor] [-i input_tensor] [-v version] [-t type]
```

表 9-19 Format 转换参数项说明

参数名	描述	是否必选
-d --dump_file	昇腾AI处理器生成的dump文件。 支持指定单个文件；单个路径；同时指定多个文件，文件名用逗号隔开，例如-d /{PATH}/dump_file1,/{PATH}/dump_file2。	是
-out --output	转换后的数据存放目录，默认为当前路径。	否
-f --format	● 命令行包含-f参数，表示进行format转换，指定转换后数据format。如果dump文件包含original_shape字段，则会根据original_shape对数据进行切片。 ● 命令行不包含-f参数，表示进行dump文件解析。	否
-s --shape	format转换需要的shape，当前仅FRACTAL_NZ转换需要配置该参数，格式为**([0-9]+,)+[0-9]+**，每个数字必须大于0。配置-f时有效。	否
-o --output_tensor	转换指定index的output数据，与-i互斥。配置-f时有效。 当-o与-i均未配置时，默认转换所有的input与output。	否
-i --input_tensor	转换指定index的input数据，与-o互斥。配置-f时有效。	否
-v --version	dump文件类型，1代表protobuf序列化后的数据文件，2代表自定义格式的数据文件。默认值为2。	否
-t --type	输出文件的类型。取值为： ● npy：输出文件保存为numpy格式。 ● msnpy：输出文件保存为numpy格式，一般用于MindSpore场景。 ● bin：输出文件保存为binary格式。 默认值为npy。	否

步骤4 调用Python，转换numpy文件为txt文件。

```
$ python3

>>> 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

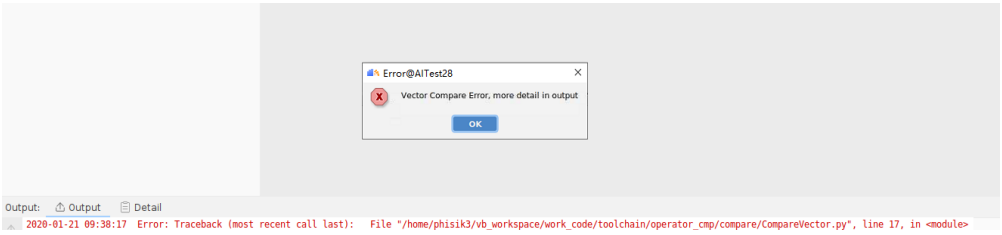
9.7.2 FAQ

9.7.2.1 Tensor 比对报错：ModuleNotFoundError: No module named 'google'

故障现象

执行Tensor比对时，界面弹出报错窗口，如图9-38所示。

图 9-38 Tensor 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，可以先使用以下命令安装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 install protobuf --user**。

9.7.2.2 如何批量处理生成的 npy 文件名异常情况

故障现象

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。

故障原因

tfdbg自身原因或运行环境原因。

处理方法

需要参考以下方法重新生成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
```

增加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 性能分析

10.1 性能分析简介

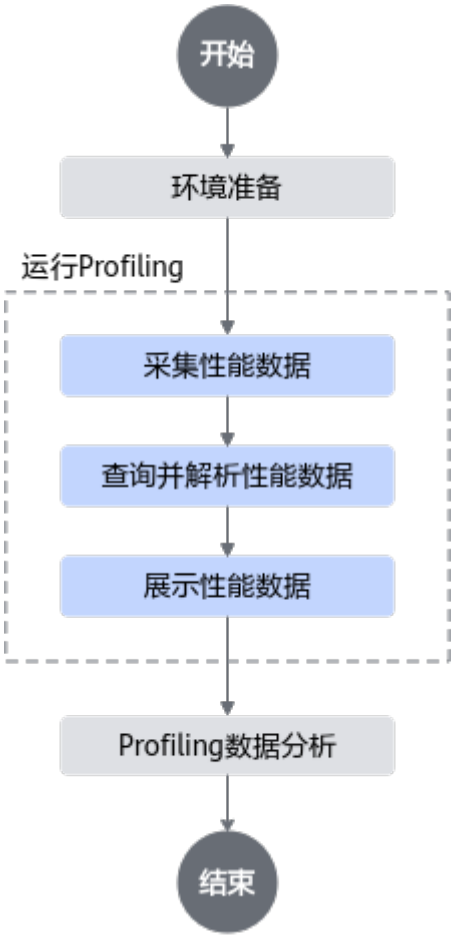
MindStudio提供了Host+Device侧丰富的性能数据采集能力和全景Timeline交互分析能力，展示了Host+Device侧各项性能指标，帮助用户快速发现和定位AI应用、芯片及算子的性能瓶颈，包括资源瓶颈导致的AI算法短板，指导算法性能提升和系统资源利用率的优化。

MindStudio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利用率和Device侧APP工程的硬件和软件性能数据。

10.2 总体流程

MindStudio提供针对硬件和软件性能数据采集、分析、汇总展示，总体流程如下：

图 10-1 总体流程



1. 环境准备。请参见**10.4 使用前准备**。
2. 运行Profiling。请参见**10.5 性能数据采集**。

   用户在配置界面开启Profiling开关（推理场景需确保APP工程可正常执行；训练场景直接执行采集）。

   a. 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向性能分析工具下发数据采集指令，由工具完成Device侧和Host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。

   b. 查询并解析性能数据。

      性能数据采集结束后，MindStudio调用性能分析工具接口查询数据。

   c. 展示性能数据。请参见**10.8 性能数据展示**。

      MindStudio通过对json文件做数据处理，生成前端展示视图数据。
3. 用户分析性能数据。请参见**10.9 性能分析样例参考**。

10.3 使用约束

- 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，需关闭数据Dump。

原因：由于Dump操作会影响系统性能，如果同时开启Profiling与Dump，会造成Profiling采集的性能数据指标不准确。如何关闭数据Dump数据请参见**准备离线模型dump数据-步骤2**设置Dump Option为None。

- Profiling工具在采集Host侧数据时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。
- Profiling工具不支持采集Ascend RC场景下的Host侧数据。
- 当前版本与历史版本的性能分析数据不一定兼容，建议不要选择历史版本的Profiling数据报告进行展示。
- 针对容器场景，性能数据采集时间建议在5min以内，同时推荐用户设置的内存大小在20G以上（例如容器配置：docker run --memory=20g *容器名*）。

10.4 使用前准备

- 推理场景请参见**应用开发**和**算子开发**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；训练场景请完成**模型训练**。
- 采集OS Runtime API和Disk时，请参见《**性能分析工具使用指南**》附录中的“安装perf、iotop、ltrace工具”和“配置用户权限”章节完成服务端Profiling工具的依赖安装和用户权限配置。

10.5 性能数据采集

操作须知

- 当前MindStudio不支持集群场景的数据采集，可通过**Import Result**导入已采集的PROF_XXX的父目录来展示集群场景性能数据。

  有关集群场景Profiling数据采集请参见《**性能分析工具使用指南**》的“附录 > 集群训练场景性能分析”章节。

- 用户通过训练工程采集数据，需要在训练工程的环境变量脚本文件env_*.sh内添加**PROFILING_OPTIONS**字段的配置信息，示例如下：

  ```
  export PROFILING_MODE=true
  export PROFILING_OPTIONS='{"output":"/tmp/profiling","training_trace":"on","task_trace":"on","fp_point":"","bp_point":"","aic_metrics":"MemoryL0"}'
  ```

  其中**output**参数指定的路径为Profiling在服务器端执行采集后输出的性能数据，最终会将性能数据拷贝到**Project Location**参数指定的路径下并生成供MindStudio展示的json结果文件。

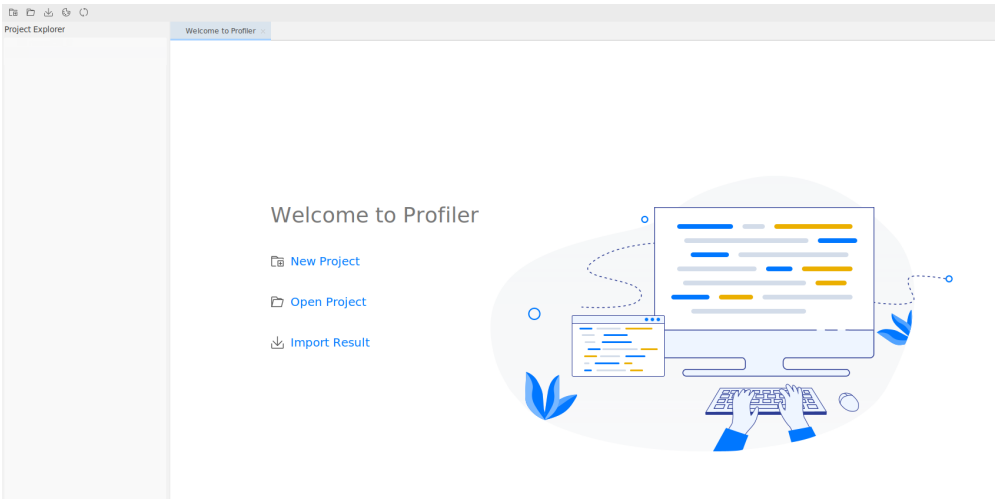
  PROFILING_OPTIONS字段为配置Profiling的采集项，请根据实际情况选择需要的参数。有关训练工程的脚本中添加Profiling配置的参数详细介绍请参见《**性能分析工具使用指南**》的“性能分析工具介绍 > 其他采集方式 > 使用TensorFlow框架接口采集 > Profiling options参数解释”。

操作步骤

步骤1 在欢迎界面的左侧导航栏单击“Projects”，单击选择并打开已编译完成的工程。

步骤2 单击菜单栏“Ascend > System Profiler”，弹出系统分析工程界面。

图 10-2 系统分析工程界面



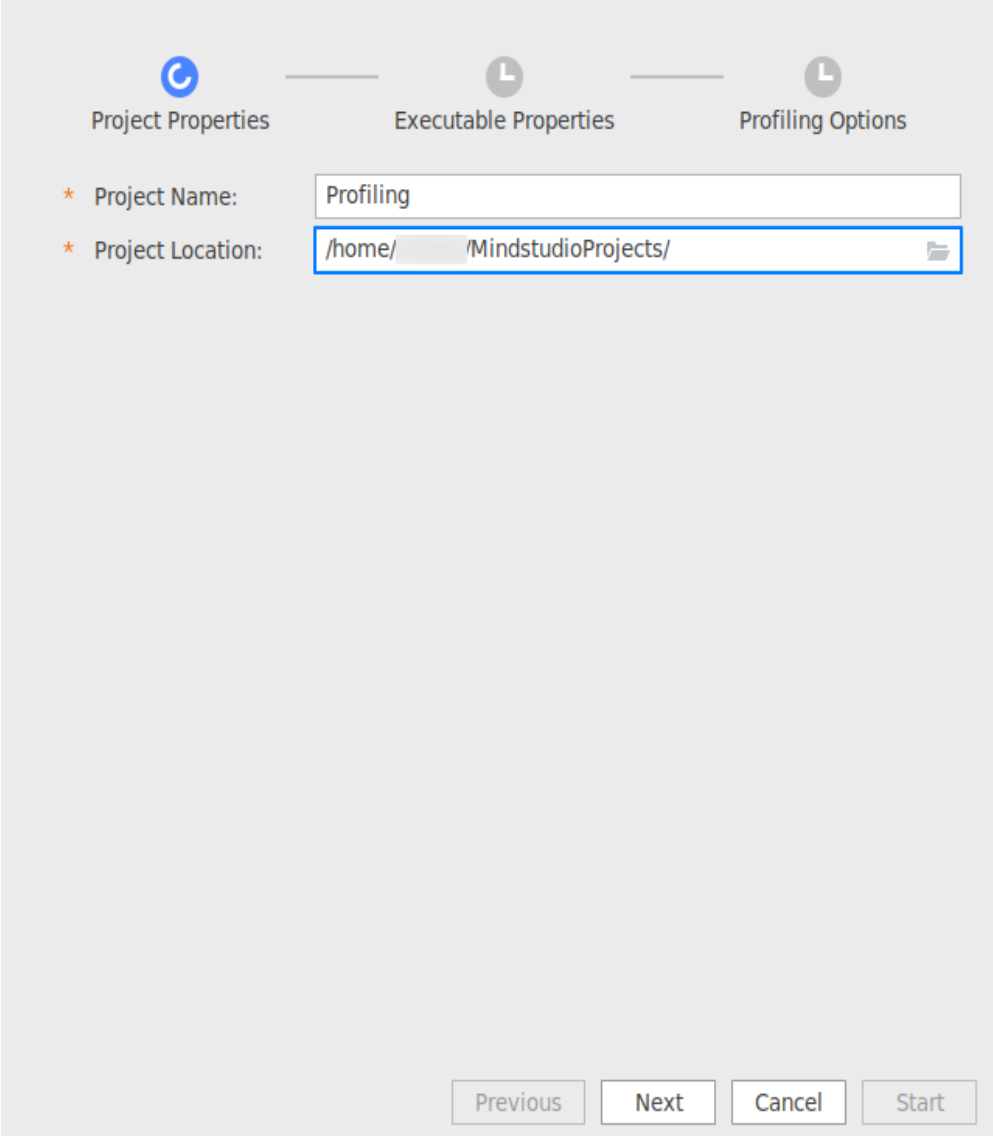
步骤3 进入系统分析工程界面，单击欢迎界面中的“New Project”或左上角的 图标，弹出Profiling配置窗口，如图10-3。

配置“Project Properties”，配置工程名称“Project Name”和选择工程路径“Project Location”。单击“Next”进入下一步。

表 10-1 Project Properties 参数说明

参数	说明
Project Name	Profiling工程名称，用户自定义。 配置后在“Project Location”设置的路径下自动创建工程名称文件夹，后续采集和解析产生的原始性能数据目录PROF_XXX和解析结果.json文件均保存在该目录下。 **说明** 解析结果.json文件命名格式为：report_{时间戳}_{device_id}_{model_id}_{iter_id}.json，其中{device_id}表示设备ID，{model_id}表示模型ID，{iter_id}表示某轮迭代的ID号。
Project Location	Profiling数据输出路径。 完成采集后在该路径下生成以“Project Name”命名的文件目录。

图 10-3 Project Properties 配置

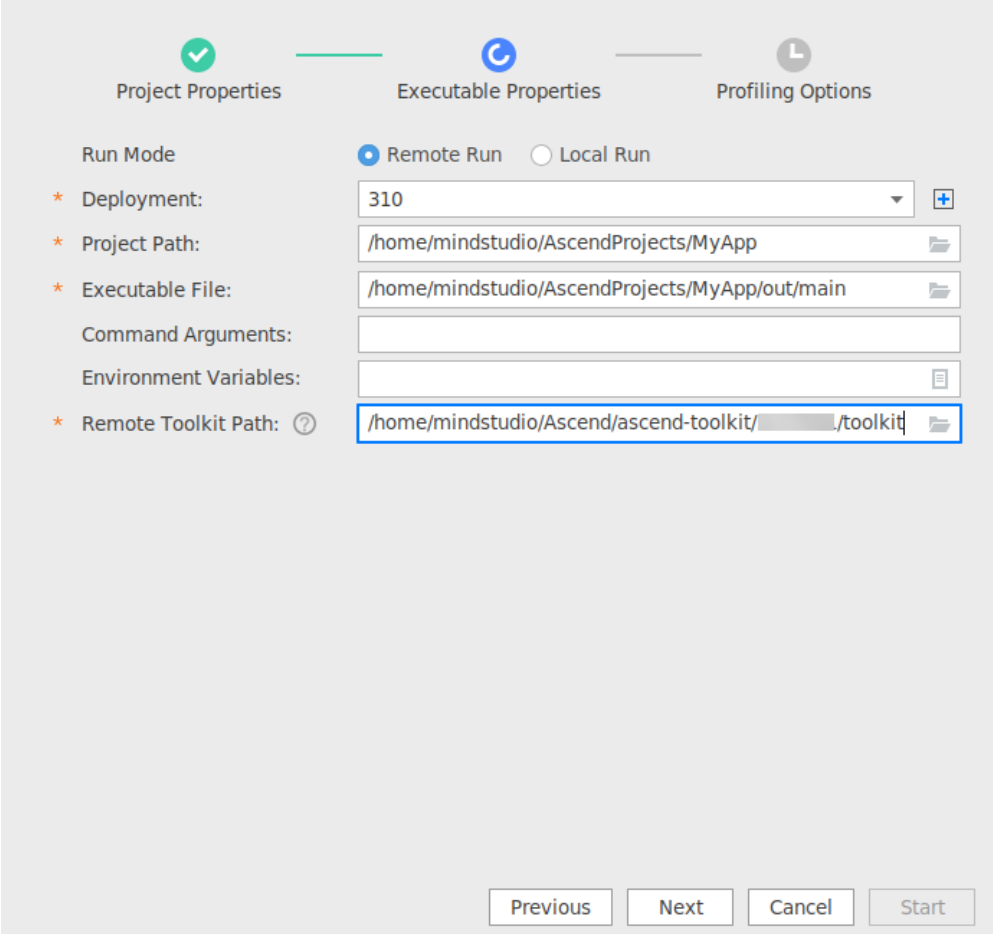


步骤4 进入“Executable Properties”配置界面，如下各图所示。

说明

- Linux使用场景下可选择Remote Run和Local Run两种模式。
- Windows使用场景下仅支持Remote Run模式，无需配置“Run Mode”。

图 10-4 Remote Run



说明

配置完成后，Deployment和对应的Environment Variables、Remote Toolkit Path参数为绑定关系，单击“Next”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则Environment Variables、Remote Toolkit Path参数自动填充，可手动修改。

图 10-5 Local Run（训练工程）

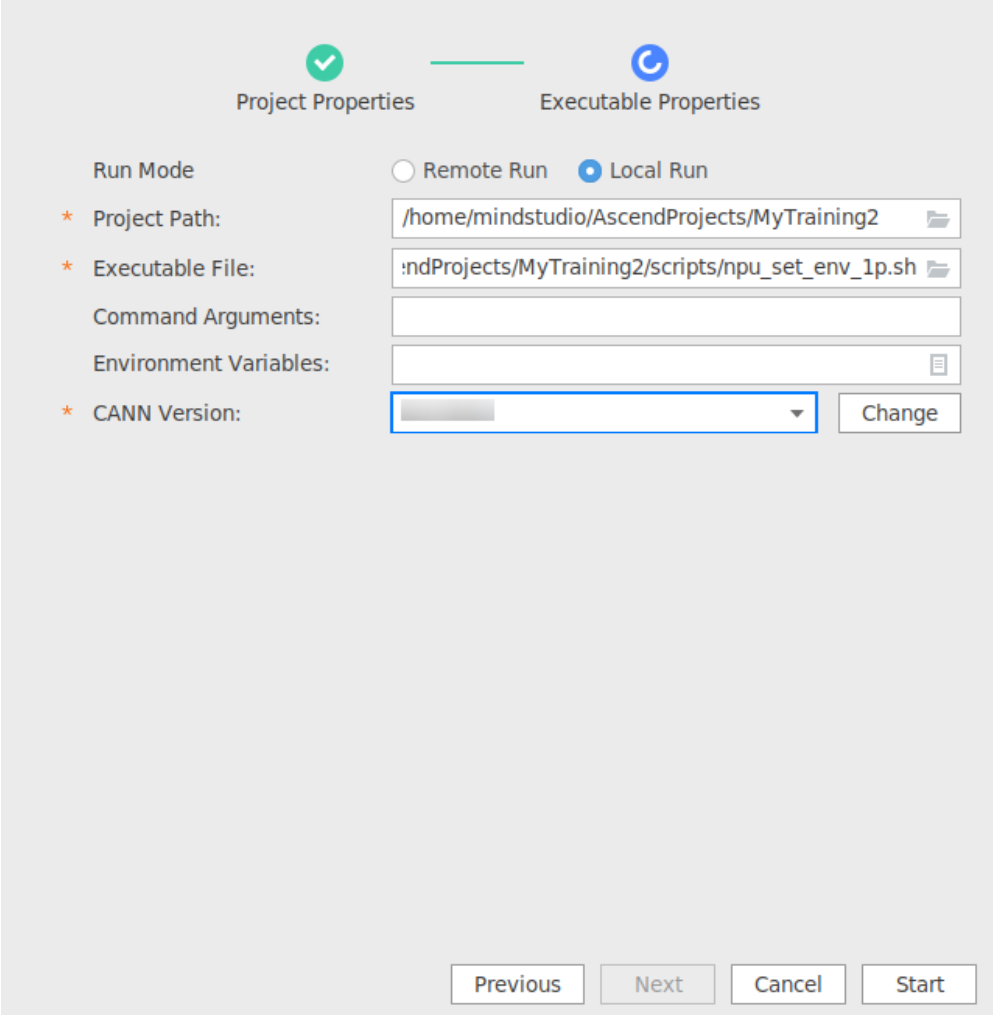


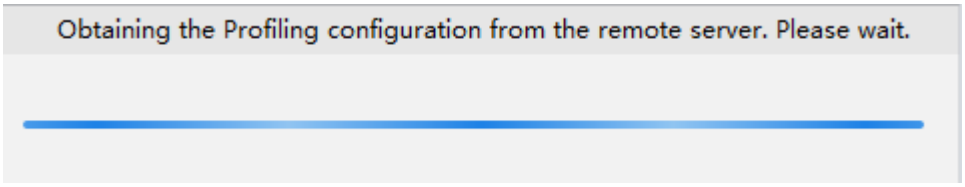
表 10-2 Executable Properties 参数说明

参数	说明
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。
Deployment	运行配置，选择Remote Run模式时可见，必选配置。通过Deployment功能，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录，具体可参见13.3 Ascend Deployment。
Project Path	执行Profiling目标工程目录，必选配置。 当指定的目标工程为训练工程时，可直接单击“Start”按钮启动Profiling。

参数	说明
Executable File	执行Profiling目标工程的可执行文件，必选配置。 需指定为Project Path子目录下的可执行文件，支持指定二进制脚本文件（如main文件）、Python脚本文件（如train.py文件）和Shell脚本文件（如npu_set_env_1p.sh文件）。 指定Python脚本文件时由于msprof工具的限制有如下要求： • pyacl工程中的Python脚本中的路径信息必须为绝对路径。 • 不支持异步接口（接口名以async结尾）的调用。 指定Shell脚本文件由用户自行提供，且无需保存在Project Path目录下。
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。
Environment Variables	环境变量配置。可直接手动配置或单击  符号，在弹出窗中配置管理。可选配置。
Remote 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，必选配置。例如配置为${HOME}/Ascend/ascend-toolkit/*{version}*/toolkit。 与Deployment参数为绑定关系，单击“Next”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则参数自动填充，可手动修改。
CANN Version	指定CANN软件包版本，选择Local Run模式时可见，必选配置。 在启动MindStudio创建工程时指定，若未指定则需单击“Change”按钮指定对应CANN软件包安装路径。

步骤5 单击“Next”后需获取Profiling configuration，如图10-6所示将出现弹框提示。

图 10-6 Obtaining Profiling configuration



步骤6 进入“Profiling Options”配置界面，其中配置AI Core Profiling时分为Task-based和Sample-based场景。如图10-7和图10-8所示。

图 10-7 Task-based 场景

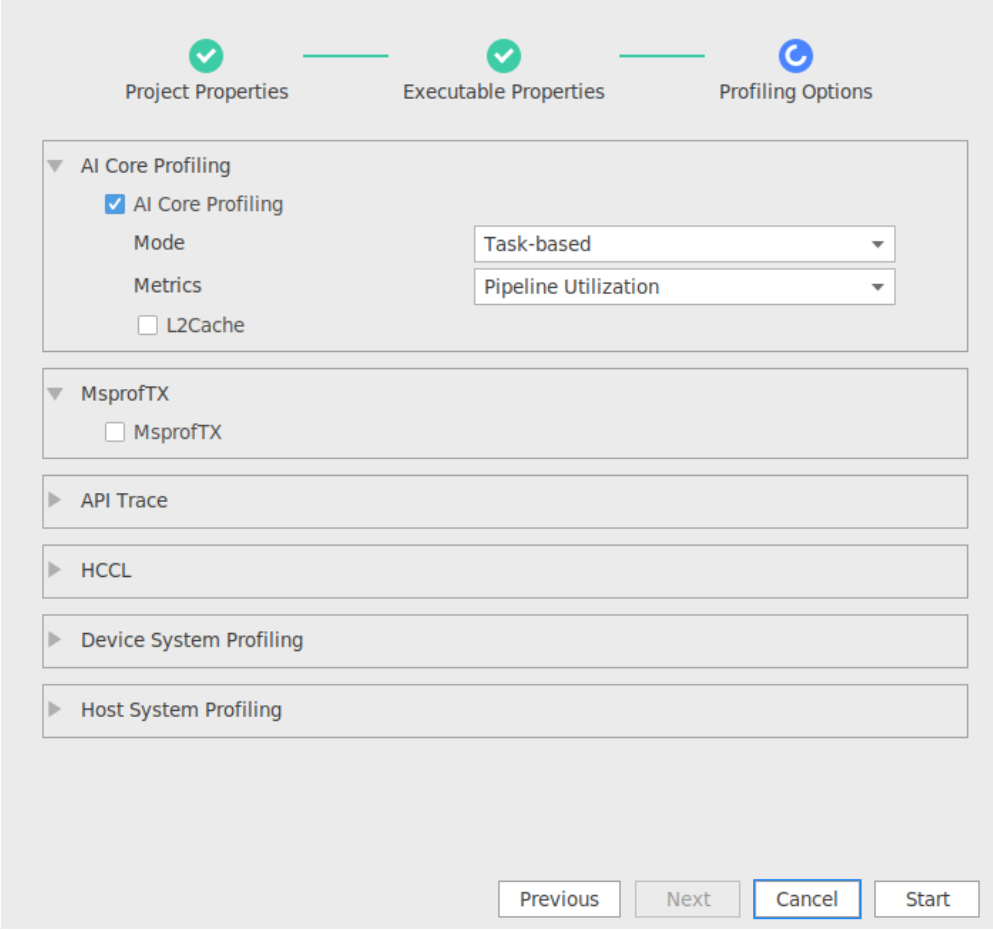


图 10-8 Sample-based 场景

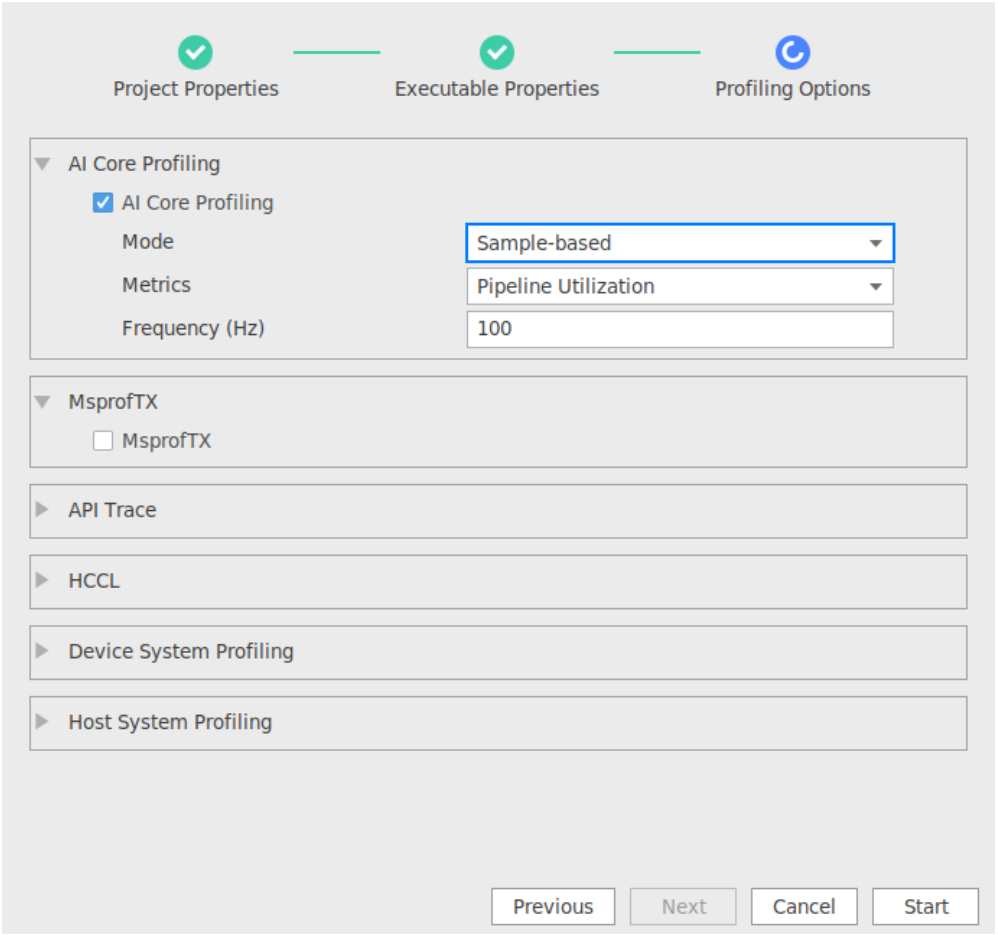


表 10-3 Profiling Options 参数说明

参数		说明
AI Core Profiling	Mode	• Task-based：AI Core采集开关，以task为粒度进行性能数据采集。默认值为Pipeline Utilization。 • 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。默认值为Pipeline Utilization。

参数		说明
	Metrics	Mode为Task-based时： ● Pipeline Utilization：采集计算单元和搬运单元耗时占比。 ● Arithmetic Utilization：cube和vector的指令类型耗时占比。 ● UB/L1/L2/Main Memory Bandwidth：UB/L1/L2/主存储器采集内存读写带宽速率。 ● L0A/L0B/L0C Memory Bandwidth：L0A/L0B/L0C采集内存读写带宽速率。 ● UB Memory Bandwidth：mte/vector/scalar采集ub读写带宽速率。 Mode为Sample-based时： ● Pipeline Utilization：采集计算单元和搬运单元耗时占比。 ● Arithmetic Utilization：cube和vector的指令类型耗时占比。 ● UB/L1/L2/Main Memory Bandwidth：UB/L1/L2/主存储器采集内存读写带宽速率。 ● L0A/L0B/L0C Memory Bandwidth：L0A/L0B/L0C采集内存读写带宽速率。 ● UB Memory Bandwidth：mte/vector/scalar采集ub读写带宽速率。
	L2Cache	Task-based场景下控制L2采样数据的开关。可选配置，默认关闭。
	Frequency(Hz)	Sample-based场景下的采集频率，取值范围为[1, 100]，默认值为100，单位Hz。
Msprof TX	MsprofTX	控制MsprofTX用户和上层框架程序输出性能数据采集的开关。可选配置，默认关闭。
API Trace	AscendCL API	AscendCL(Ascend Compute Language)采集开关，采集接口流水信息。默认开启。
	Runtime API	Runtime采集开关，采集运行管理器接口流水信息。可选配置，默认关闭。
	Graph Engine(GE)	Graph Engine采集开关，采集模型图引擎调度流水信息。默认开启，不可关闭。
	AICPU Operators	AI CPU采集开关，采集aicpu数据增强的Profiling数据。可选配置，默认关闭。

参数			说明
HCCL	HCCL		控制HCCL数据采集开关。可选配置，默认关闭。 当前采集完成后，默认导出对应迭代数目最多模型号（Model ID）的第一轮迭代数据。
Device System Profiling	CPU & Memory Usage Profiling		系统CPU usage及System memory采集开关。可选配置，默认关闭。 可以更改采样频率Frequency(Hz)，取值范围为[1, 10]，默认频率为10Hz。
Host System Profiling	Application Based System Profiling	CPU	采集Host侧CPU资源利用率信息。可选配置，默认关闭。
		Memory	采集Host侧Memory资源利用率信息。可选配置，默认关闭。
		Disk	采集Host侧Disk资源利用率信息。可选配置，默认关闭。 **说明** 采集Disk调用数据需要安装第三方开源工具iotop。参见**10.4 使用前准备**。
		Network	采集Host侧Network资源利用率信息。可选配置，默认关闭。
		Syscall & PThreadcall	Host侧syscall和pthreadcall数据。可选配置，默认关闭。
	System CPU & Memory Usage	CPU	采集Host侧系统和所有进程的CPU资源利用率信息。可选配置，默认关闭。
		Memory	采集Host侧系统和所有进程的Memory资源利用率信息。可选配置，默认关闭。
		Frequency(Hz)	CPU利用率和内存利用率的采集频率，取值范围为[1,50]，默认值50，单位hz。

说明

表3 Profiling Options参数说明为全量采集配置项，芯片的实际支持情况请以界面展示为准。

步骤7 完成上述配置后单击窗口右下角的“Start”按钮，启动性能数据采集。

工程执行完成后，MindStudio自动弹出性能分析结果视图。

----结束

10.6 Import Result

本功能的作用是对采集后的性能原始数据通过MindStudio界面进行解析并展示。配置步骤如下：

步骤1 单击顶部菜单栏“Ascend > System Profiler”，弹出系统分析工程界面。

步骤2 进入系统分析工程界面，单击欢迎界面中的“Import Result”或左上角的  图标，弹出“Import Profile Data”配置窗口 。

步骤3 配置Import Result的数据源输入路径、数据输出路径以及输出展示的工程名。如**图10-9**所示。

图 10-9 Import Result 配置

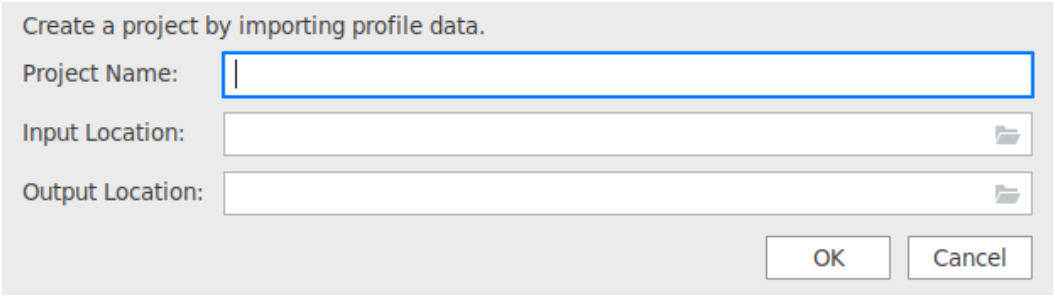


表 10-4 Import Result 参数说明

参数	说明
Project Name	输出展示的工程名，用户自定义。
Input Location	数据源输入路径。指定为msprof命令指定的数据输出路径（即PROF_XXX目录的父目录）或直接指定为PROF_XXX目录。
Output Location	数据输出路径。

说明

- 有关msprof命令的详细使用请参见《**性能分析工具使用指南**》中的“性能分析工具介绍 > msprof命令行工具”章节。
- msprof命令在多device的环境下执行时，数据输出路径下会生成对应device数量的PROF_XXX文件。
- msprof命令在相同路径下执行多次性能数据采集后，数据输出路径下会生成多个PROF_XXX文件。
- 当.json数据文件大小超出了Import Result自动展示结果的最大缓存，可能无法自动展示，请手动打开文件。
- 支持导入训练集群场景的Profiling数据，即将待展示的所有PROF_XXX目录汇总到指定目录，“Input Location”配置为该目录，详细展示界面请参见**10.8.3 Analysis Summary**和**10.8.6 集群场景数据**。
- 有关集群场景性能数据的采集请参见《**性能分析工具使用指南**》的“附录 > 集群训练场景性能分析”章节。
- MindStudio加载性能数据时如果提示“Low memory”，请参见**14.3.26 加载集群场景性能数据时提示“Low memory”**处理。

----结束

10.7 性能分析工程管理

创建新工程

步骤1 在欢迎界面的左侧导航栏单击“Projects”，单击选择并打开已编译完成的工程。

步骤2 单击菜单栏“Ascend > System Profiler”，弹出系统调优工程界面。

步骤3 进入系统分析工程界面，单击欢迎界面中的“New Project”或左上角的  图标。

步骤4 参见**步骤3**到**步骤7**完成创建。

----结束

打开未展示的工程

步骤1 在欢迎界面的左侧导航栏单击“Projects”，单击选择并打开已编译完成的工程。

步骤2 单击菜单栏“Ascend > System Profiler”，弹出系统分析工程界面。

步骤3 进入系统分析工程界面后，以下方式任选一种对未展示的工程目录进行选择。

- 在欢迎界面中单击“Open Project”。
- 单击左上角的  图标。
- 右键单击左侧导航栏中空白处，选择“Open…”。

步骤4 单击“OK”打开工程。

----结束

打开工程

说明

数据采集执行成功后，自动进行性能分析的结果展示。

步骤1 进入系统分析工程界面后，右键单击左侧导航栏中对应的工程目录。

步骤2 在弹出菜单中单击“Open Project”。

----结束

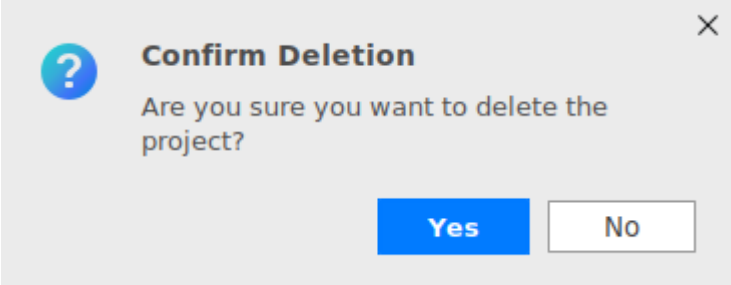
删除工程

步骤1 进入系统分析工程界面后，右键单击左侧导航栏中对应的工程目录。

步骤2 在弹出菜单中单击“Delete Project”。

步骤3 弹出如**图10-10**所示确认框，单击“Yes”。

图 10-10 删除工程



----结束

刷新导航栏

单击系统分析工程界面左上角的 图标。刷新后可以将更新导航栏状态，展示当前最新的工程目录树。

10.8 性能数据展示

10.8.1 展示视图介绍

性能数据采集执行成功后，自动展示数据结果，如下图所示。

图 10-11 性能分析视图窗口（推理）

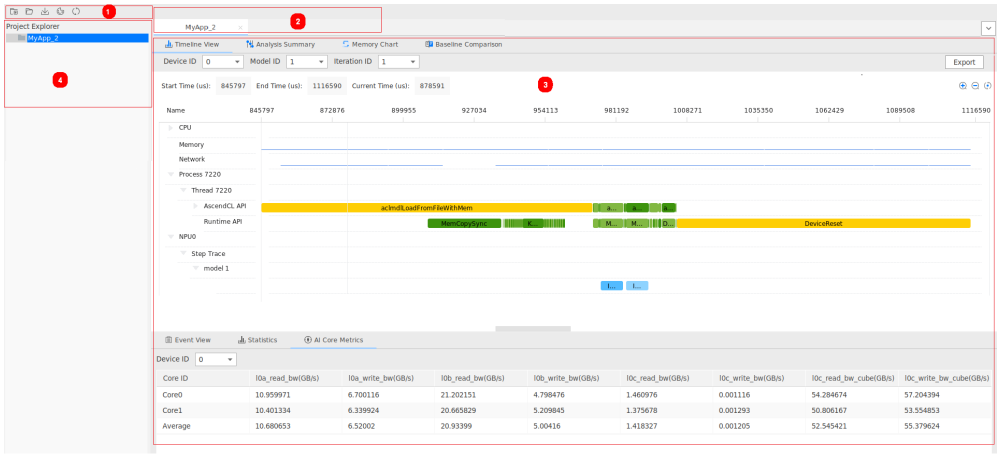
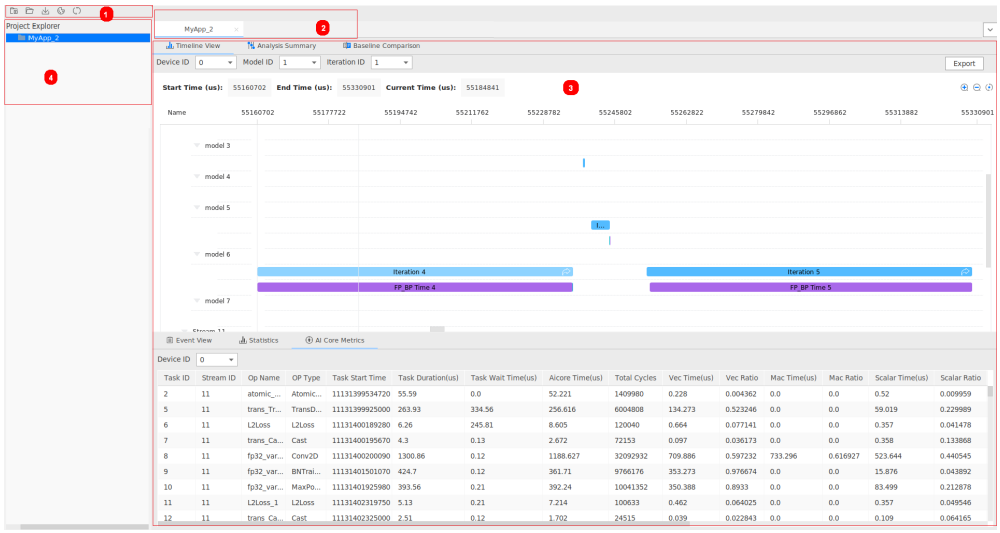


图 10-12 性能分析视图窗口（训练）



说明

- 光标悬浮在视图上的各个时间线上时，可显示对应接口的运行信息，包括接口的启动时间（Start Time）、停止时间（End Time）、执行持续时间（Duration）和接口名（Name）等。有关更多接口的字段含义请参见《**性能分析工具使用指南**》中的“性能数据文件参考”章节。
- 性能数据采集后生成的.json数据文件命名格式为：report_{时间戳}_{device_id}_{model_id}_{iter_id}.json，其中{device_id}表示设备ID，{model_id}表示模型ID，{iter_id}表示某轮迭代的ID号。
- 训练场景采集迭代轨迹数据、AI Core数据和算子数据等，不配置**步骤6**，即不进行“Profiling Options”各项数据采集，因此**图10-12**界面展示的数据为**图10-11**界面展示数据的子集。
- 当性能数据采集后生成的.json数据文件大小超出了Profiling自动展示结果的最大缓存，可能无法自动展示，请手动打开文件。

性能分析视图窗口分为如下四个区域：

区域1：菜单栏。从左到右分别为New Project（创建新工程）、Open Project（打开未展示的工程）、Import Result（导入原始数据并解析和展示）、Timeline Color（打开Timeline颜色配置）和Refresh（刷新导航栏）五项功能。详细介绍请参见**10.7 性能分析工程管理**和**10.8.2.1 Timeline颜色配置**。

区域2：展示打开Profiling工程项目，单击对应页签则显示对应工程的结果。

区域3：以视图方式展示Profiling采集结果，包括：**10.8.2 Timeline View**、**10.8.3 Analysis Summary**、**10.8.4 Memory Chart**（仅推理）、**10.8.5 Baseline Comparison**、**10.8.6.1 Cluster Iteration Analysis**（集群）、**10.8.6.2 Data Preparation**（集群）、**10.8.6.3 Communication Analysis**（集群）、**10.8.7 Host System Analysis**。

区域4：导航栏。显示用户指定的工程目录。提供历史数据管理功能，包括：Open Project（打开报告）、Delete Project（删除报告）等；支持在空白处右键New Project（创建新工程）、Open…（打开未展示的工程）、Refresh（刷新导航栏）。详细介绍请参见**10.7 性能分析工程管理**。

Profiling支持Task-Based和Sample-Based两种不同方式的AI Core采集方式，因此会出现Profiling各页签展示结果不同。

表 10-5 AI Core 采集方式视图差异

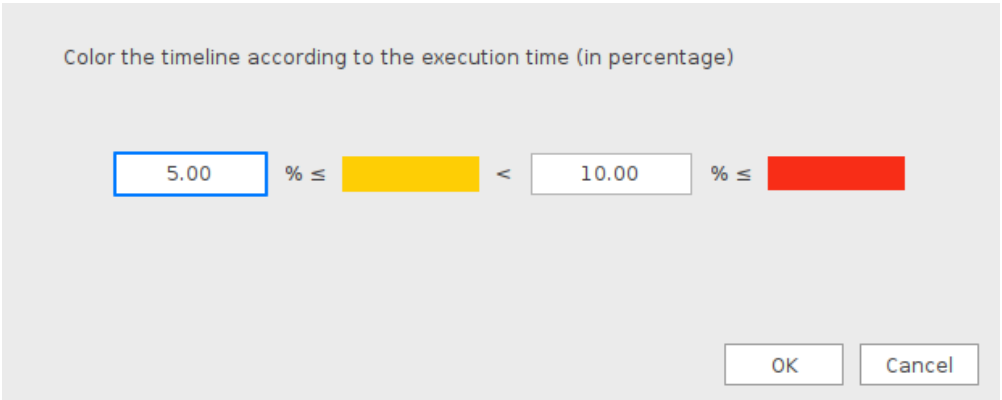
视图	Task-Based	Sample-Based
AI Core Utilization（Analysis Summary）	不展示	展示
Timeline View	展示	展示
CPU	展示	展示
Memory	展示	展示
Disk	展示	展示
Network	展示	展示
Os Runtime	展示	展示
ACL API	展示	展示
Runtime API	展示	展示
GE	展示	展示
MsprofTX	展示	展示
Step Trace	展示	展示
AI Core task	展示	不展示
AI CPU task	展示	展示
HCCL	展示	展示

10.8.2 Timeline View

10.8.2.1 Timeline 颜色配置

在顶部菜单栏选择“Ascend > System Profiler”，进入系统分析工程界面，单击左上角的  图标进入Timeline颜色配置窗口。如图10-13所示。

图 10-13 Timeline 颜色配置



可以根据API执行时间自定义配置Timeline中的颜色显示比例，显示格式为：绿色 a% ≤ 黄色 < b% ≤ 红色，此设置的数值表示在Profiling采集过程中API运行时间的耗时占比。

例如图10-13中所示，默认配置为绿色 5% ≤ 黄色 < 10% ≤ 红色，则耗时占比小于等于5%的时间线显示为绿色，耗时占比在5%~10%的时间线显示为黄色，耗时占比大于等于10%的时间线显示为红色。

说明

配置颜色比例的数值最多可精确到小数点后两位。

10.8.2.2 Timeline 视图

Timeline View包含左侧导航窗格、右侧图形化窗格和下方数据窗格，如下图所示：

图 10-14 Timeline View

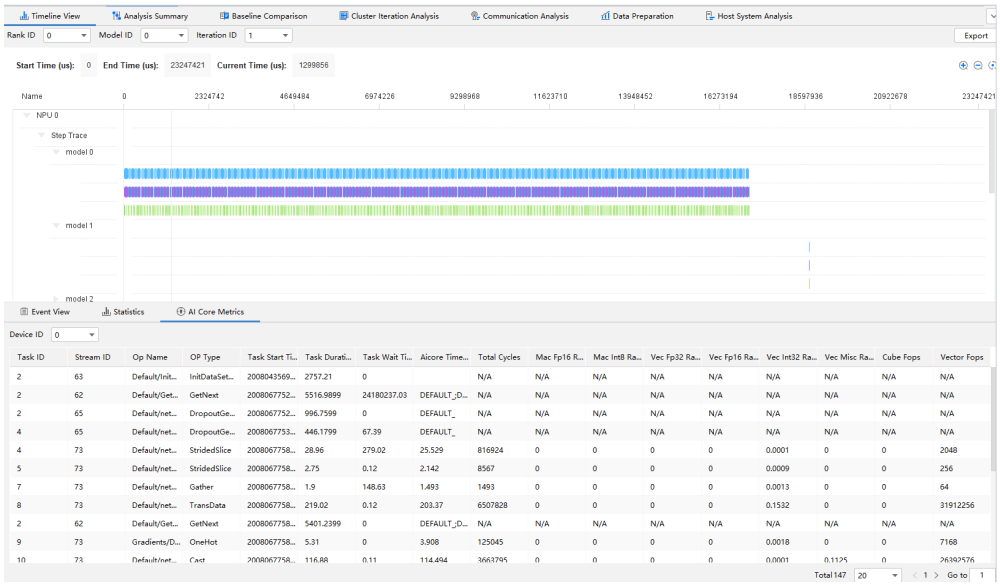


图 10-15 Timeline View（PyTorch 集群场景）

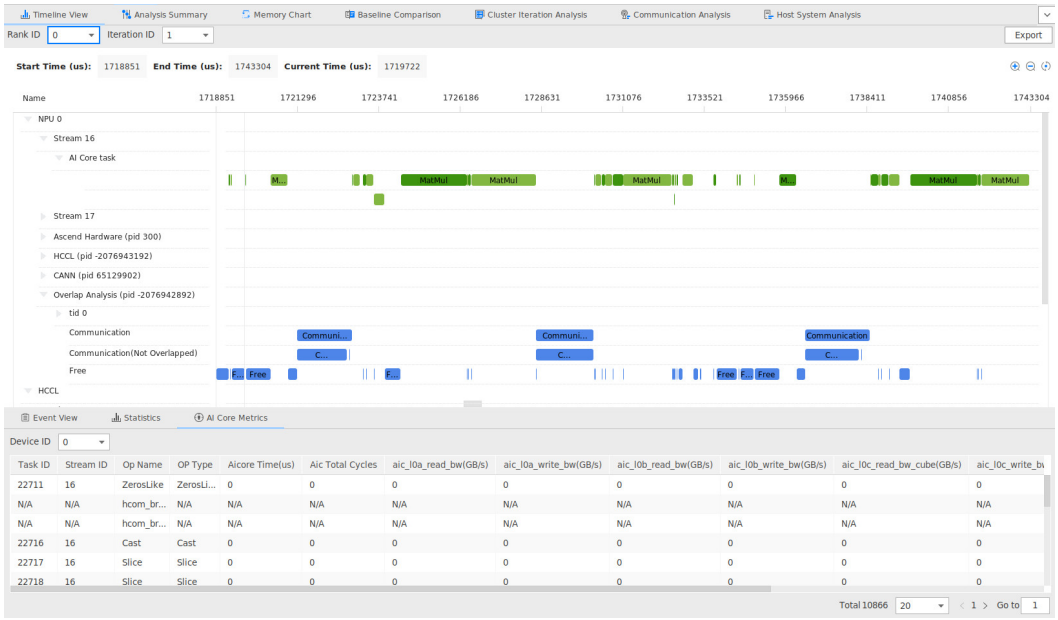
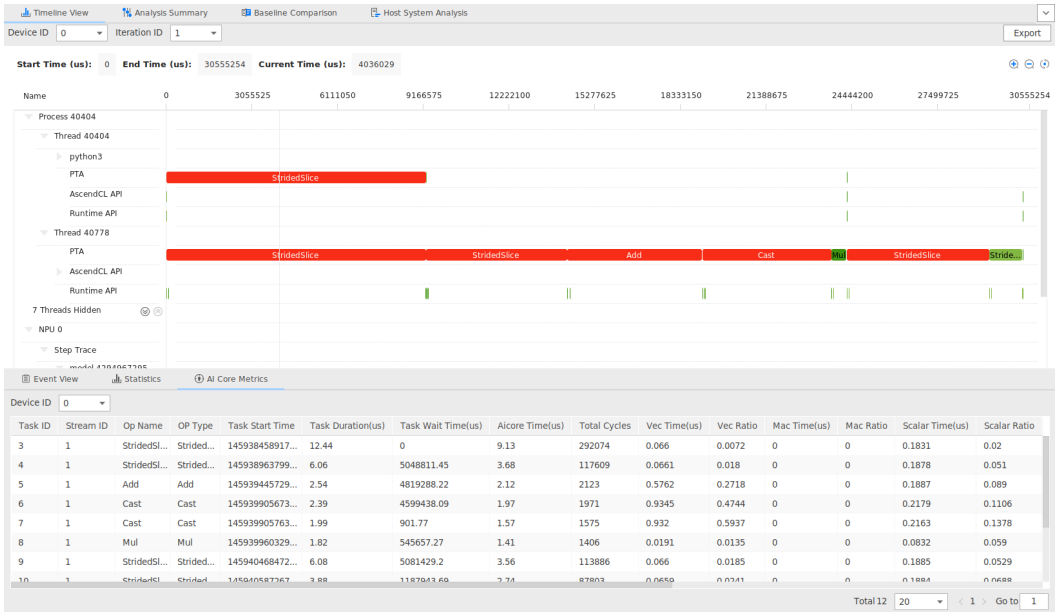


图 10-16 Timeline View（PyTorch E2E Profiling 场景）



- 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。
- 数据窗格以表格的形式呈现性能分析工具采集之后的各项数据，分为Event View、Statistics和AI Core Metrics。

说明

- 性能分析数据中涉及到的时间节点（非Timestamp）为系统单调时间只与系统有关，非真实时间。
- “Start Time”和“End Time”表示本次性能采集数据的展示时间范围。
- “Current Time”表示光标所在时间块的开始时间。
- 将光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。
- 如果同一线程中有多个OS Runtime API同时执行，会分行显示。
- 如果同一Stream中有多个AI Core task同时执行，会分行显示。
- 当选中某个时间点时，按住Ctrl键并向上/下滚动鼠标滑轮或单击视图右上角的 / ，实现Timeline视图的放大/缩小，单击视图右上角的 实现视图复位。
- 当选中某个时间点时，可通过左右拖拉鼠标选择时间长度。以Current Time(us)为界点，鼠标左右拖动显示选中的时间范围。
- 集群或多Device场景下的性能数据，默认导出最小的Rank ID/Device ID所对应迭代数目最多的模型号（Model ID）的第一轮迭代数据。
- PyTorch框架下的性能数据，默认导出最小Rank ID/Device ID所对应的第一轮迭代数据，如图10-15或图10-16所示。

在Profiling工程执行完成后，以时序图的呈现方式为用户提供全流程推理/训练过程中的运行情况，按照调度流程来呈现整体的运行状况。实际展示情况与Profiling采集时的所选项以及设备有关，请以设备实际情况为准。

如下表格中按展示顺序介绍各字段。

表 10-6 硬件信息

字段名	字段解释
CPU	CPU。
Memory	内存。
Disk	磁盘。
Network	网络带宽。
Start Time	CPU、内存、磁盘和网络带宽的进程开始时间，单位us。
End Time	CPU、内存、磁盘和网络带宽的进程结束时间，单位us。
Duration	CPU、内存、磁盘和网络带宽的运行耗时，单位us。
Usage	CPU、内存、磁盘和网络带宽的利用率。
注：光标悬浮在Timeline上时展示Start Time、End Time、Duration和Usage。	

表 10-7 各个组件的耗时数据

字段名	字段解释
Process {ID}	进程ID。
Thread {ID}	线程ID。
MsprofTX	MsprofTX性能数据。
Os Runtime	展示每个线程调用Os Runtime的时序信息。
PTA	PyTorch层算子异步下发的task queue队列的时序信息。
AscendCL API	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。
ACL_RTS	RTS类型的AscendCL API。
ACL_MODEL	MODEL类型的AscendCL API。
Runtime API	展示每个线程调用runtime API的时序信息。
GE	展示模型的数据输入、推理、数据输出耗时。
Start Time	接口开始运行的时间，单位us。
End Time	接口结束运行的时间，单位us。
Duration	当前接口调用耗时，单位us。
Name	API名。
注：光标悬浮在Timeline上时展示Start Time、End Time、Duration和Name。	

表 10-8 昇腾 AI 处理器系统数据

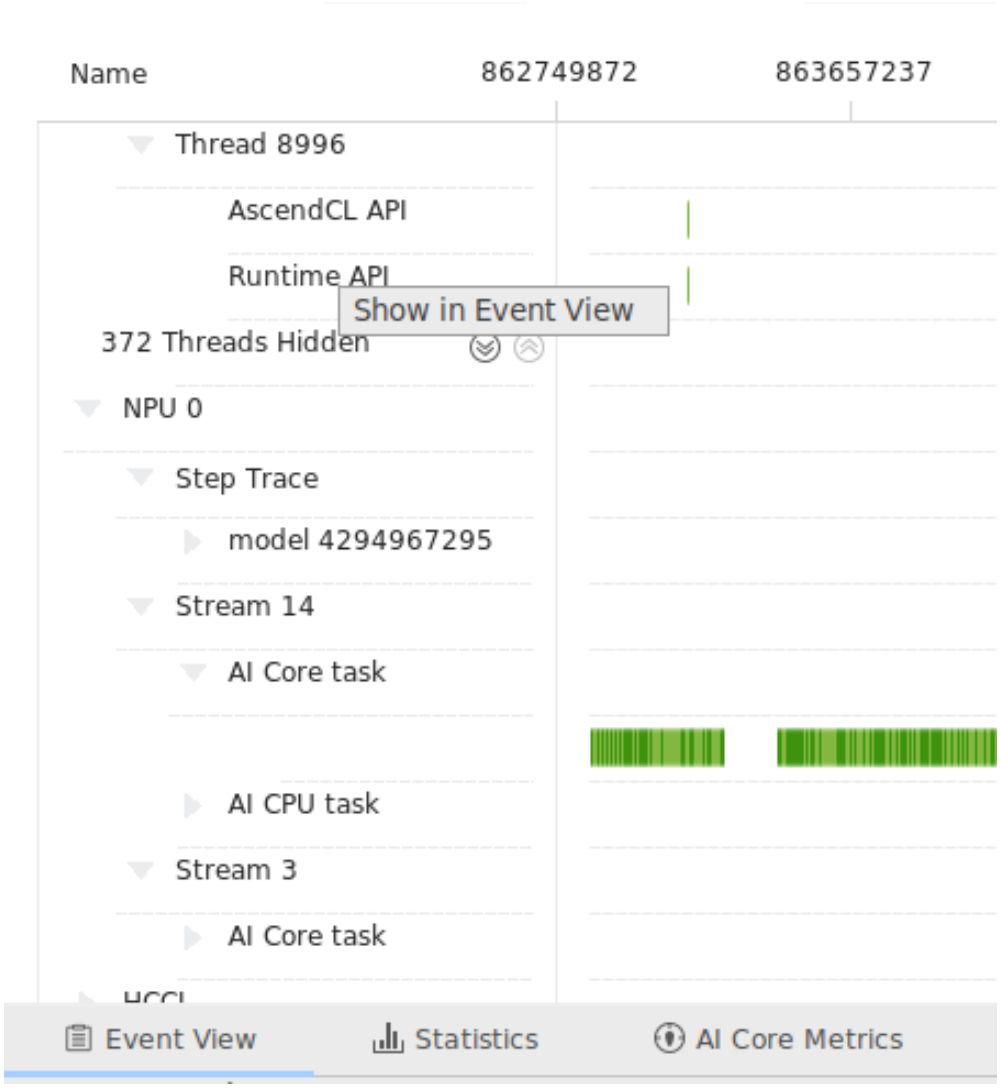
字段名	字段解释
NPU {ID}	昇腾AI处理器的编号。
Step Trace	迭代轨迹数据，每轮迭代的耗时。
model {ID}	模型ID，在Step Trace下按顺序展示。 以下任意一种方式可以导出并展示某个Model下的某轮迭代数据。 • 单击某个Model下的某轮迭代（Iteration）的  按钮，弹出提示框后单击“Yes”导出。 • 在界面左上角选择Device ID、Model ID和Iteration ID，单击“Export”导出。
Name	接口名。
Iteration ID	迭代ID。

字段名	字段解释
FP Start	FP开始时间，单位us。
Iteration End	每轮迭代结束的时间，单位us。
Iteration Time	迭代时长，单位us。
Stream {ID}	Stream任务的ID。
AI Core task	展示每个Stream的AI Core task时序信息。
AI CPU task	展示每个Stream的AI CPU task时序信息。
Other task	展示每个Stream的Other task时序信息。
Start Time	AI Core task、AI CPU task和Other task开始运行的时间，单位us。
End Time	AI Core task、AI CPU task和Other task结束运行的时间，单位us。
Duration	AI Core task、AI CPU task和Other task调用耗时，单位us。
Status	AI Core task、AI CPU task和Other task的运行状态。
Task Type	AI Core task、AI CPU task和Other task的任务类型。
Stream ID	AI Core task、AI CPU task和Other task的stream ID。
Op Name	算子名。
Task ID	AI Core task、AI CPU task和Other task的task ID。
注：光标悬浮在Timeline上时展示Name、Iteration ID、FP Start、Iteration End、Iteration Time、Start Time、End Time、Duration、Status、Task Type、Stream ID、Op Name和Task ID。	

10.8.2.3 Event 视图

在Timeline View界面的左侧导航窗格中右键标签名，选择“Show in Event View”，可以在下方数据窗格的Event View中看到对应Timeline的顺序执行信息，如图10-17所示。

图 10-17 Show in Event View



可查看项目包括：

- Process > Thread > AscendCL API（ACL_MODEL/ACL_RTS）
- Process > Thread > Runtime API
- Process > Thread > Os Runtime
- NPU > Stream > AI Core task

AscendCL API（ACL_MODEL/ACL_RTS）

图 10-18 AscendCL API（ACL_MODEL/ACL_RTS）

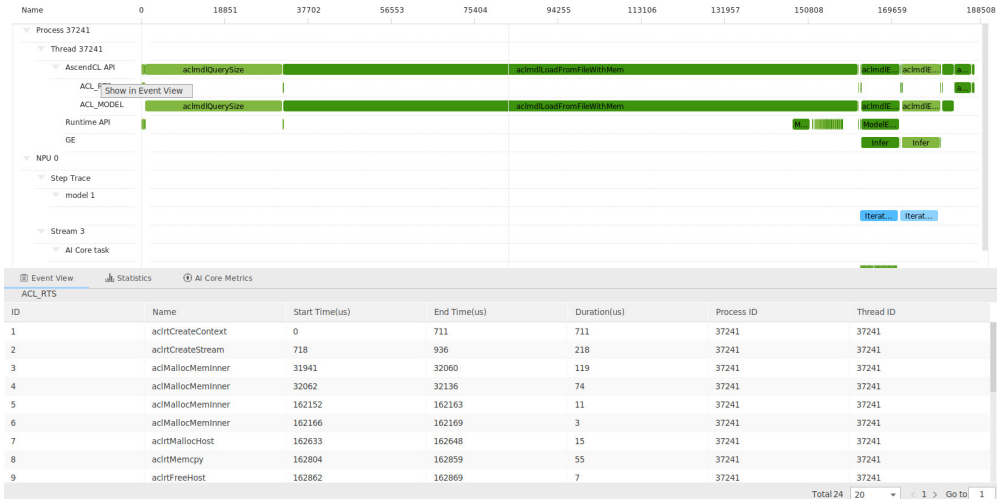
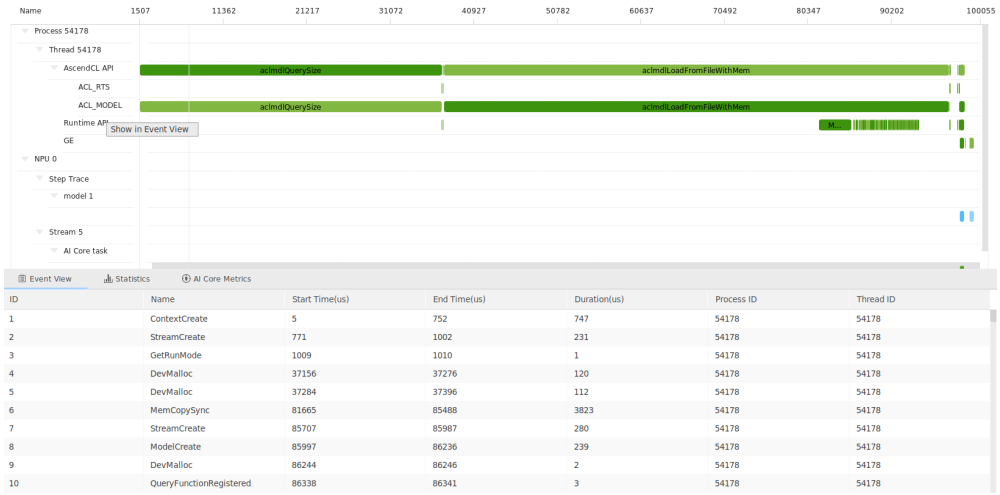


表 10-9 字段说明（ACL_MODEL/ACL_RTS）

字段名	字段解释
ID	AscendCL API的ID。
Name	AscendCL API的名称。
Start Time(us)	AscendCL API的开始运行时间，单位为us。
End Time(us)	AscendCL API的结束运行时间，单位为us。
Duration(us)	AscendCL API的运行耗时，单位为us。
Process ID	AscendCL API对应进程ID。
Thread ID	AscendCL API对应线程ID。

Runtime API

图 10-19 Runtime API



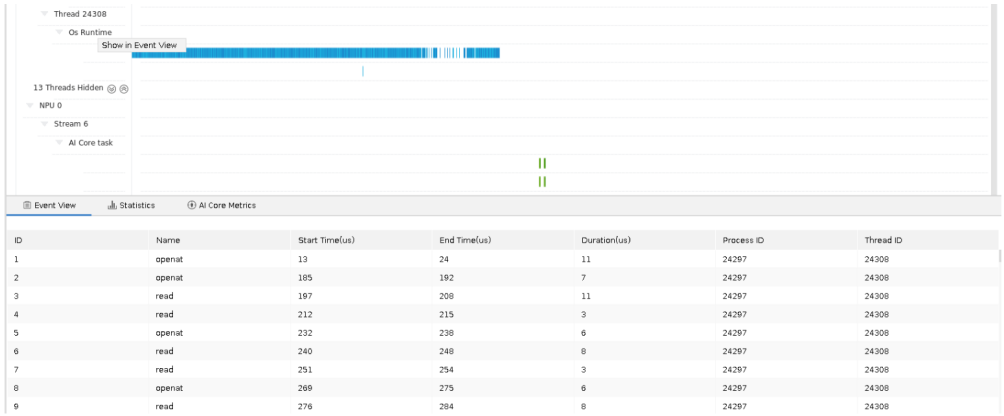
ID	Name	Start Time(us)	End Time(us)	Duration(us)	Process ID	Thread ID
1	ContextCreate	5	752	747	54178	54178
2	StreamCreate	771	1002	231	54178	54178
3	GetRunMode	1009	1010	1	54178	54178
4	DevMalloc	37156	37276	120	54178	54178
5	DevMalloc	37284	37396	112	54178	54178
6	MemCopySync	81665	85488	3823	54178	54178
7	StreamCreate	85707	85987	280	54178	54178
8	ModelCreate	85997	86236	239	54178	54178
9	DevMalloc	86244	86246	2	54178	54178
10	QueryFunctionRegistered	86338	86341	3	54178	54178

表 10-10 字段说明

字段名	字段解释
ID	Runtime API的ID。
Name	Runtime API的名称。
Start Time(us)	Runtime API的开始运行时间，单位为us。
End Time(us)	Runtime API的结束运行时间，单位为us。
Duration(us)	Runtime API的运行耗时，单位为us。
Process ID	Runtime API对应进程ID。
Thread ID	Runtime API对应线程ID。

Os Runtime

图 10-20 Os Runtime



ID	Name	Start Time(us)	End Time(us)	Duration(us)	Process ID	Thread ID
1	openat	13	24	11	24297	24308
2	openat	185	192	7	24297	24308
3	read	197	208	11	24297	24308
4	read	212	215	3	24297	24308
5	openat	232	238	6	24297	24308
6	read	240	248	8	24297	24308
7	read	251	254	3	24297	24308
8	openat	269	275	6	24297	24308
9	read	276	284	8	24297	24308

表 10-11 字段说明

字段名	字段解释
ID	Os Runtime的ID。
Name	Os Runtime的名称。
Start Time(us)	Os Runtime的开始运行时间，单位为us。
End Time(us)	Os Runtime的结束运行时间，单位为us。
Duration(us)	Os Runtime的运行耗时，单位为us。
Process ID	Os Runtime对应进程ID。
Thread ID	Os Runtime对应线程ID。

AI Core task

图 10-21 AI Core task

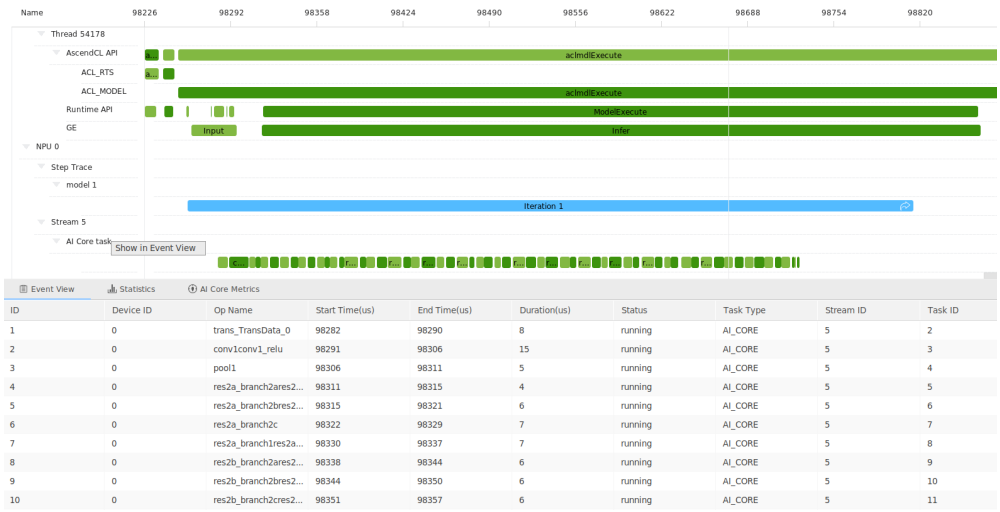


表 10-12 字段说明

字段名	字段解释
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 Time(us)	AI Core task的开始运行时间，单位为us。
End Time(us)	AI Core task的结束运行时间，单位为us。
Duration(us)	AI Core task的运行耗时，单位为us。

字段名	字段解释
Status	AI Core task的运行状态。
Task Type	AI Core task的任务类型。
Stream ID	AI Core task的stream ID。
Task ID	AI Core task的task ID。

10.8.2.4 Statistics 视图

单击下方数据窗格的Statistics，通过左上角下拉框选择不同类别的数据，可以查看API Statistic、AscendCL API、OS Runtime API、Runtime API、OPs和Op Info调用情况数据。

说明

若使用当前版本采集性能数据，AscendCL API、OS Runtime API、Runtime API无数据展示，相关数据将汇总在API Statistic进行展示。

API Statistic

图 10-22 API Statistic

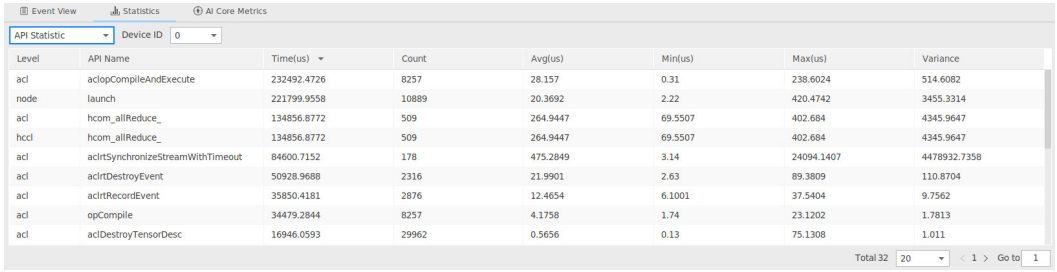


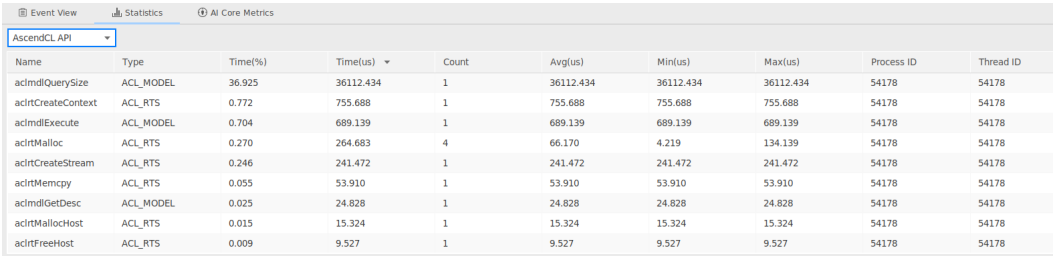
表 10-13 字段说明

字段名	字段解释
Level	API所属层级，包含AscendCL、Runtime、Node、Model、HCCL层级的API。
API Name	对应调用API名称。
Time(us)	对应调用API时间长度，单位为us。 可以单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。
Min(us)	对应API单次调用最短时间，单位为us。
Max(us)	对应API单次调用最长时间，单位为us。

字段名	字段解释
Variance	对应API单次调用的耗时方差。

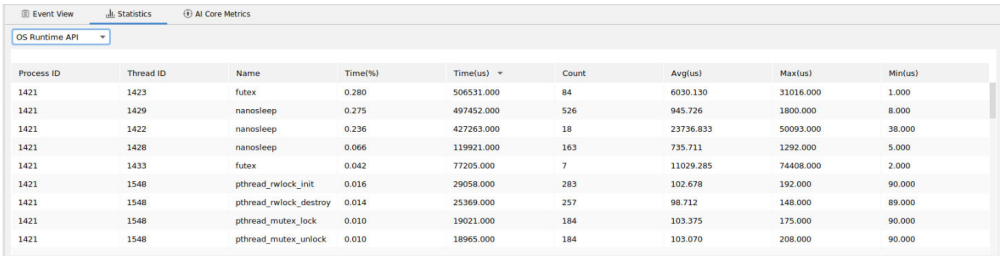
AscendCL API 和 OS Runtime API

图 10-23 AscendCL API



Name	Type	Time(%)	Time(us)	Count	Avg(us)	Min(us)	Max(us)	Process ID	Thread ID
aclmdlQuerySize	ACL_MODEL	36.925	36112.434	1	36112.434	36112.434	36112.434	54178	54178
aclrtCreateContext	ACL_RTS	0.772	755.688	1	755.688	755.688	755.688	54178	54178
aclmdlExecute	ACL_MODEL	0.704	689.139	1	689.139	689.139	689.139	54178	54178
aclrtMalloc	ACL_RTS	0.270	264.683	4	66.170	4.219	134.139	54178	54178
aclrtCreateStream	ACL_RTS	0.246	241.472	1	241.472	241.472	241.472	54178	54178
aclrtMemcpy	ACL_RTS	0.055	53.910	1	53.910	53.910	53.910	54178	54178
aclmdlGetDesc	ACL_MODEL	0.025	24.828	1	24.828	24.828	24.828	54178	54178
aclrtMallocHost	ACL_RTS	0.015	15.324	1	15.324	15.324	15.324	54178	54178
aclrtFreeHost	ACL_RTS	0.009	9.527	1	9.527	9.527	9.527	54178	54178

图 10-24 OS Runtime API



Process ID	Thread ID	Name	Time(%)	Time(us)	Count	Avg(us)	Max(us)	Min(us)
1421	1423	futex	0.280	506531.000	84	6030.130	31016.000	1.000
1421	1429	nanosleep	0.275	497452.000	526	945.726	1800.000	8.000
1421	1422	nanosleep	0.236	427263.000	18	23736.833	50093.000	38.000
1421	1428	nanosleep	0.066	119921.000	163	735.711	1292.000	5.000
1421	1433	futex	0.042	77205.000	7	11029.285	74408.000	2.000
1421	1548	pthread_rwlock_init	0.016	29058.000	283	102.678	192.000	90.000
1421	1548	pthread_rwlock_destroy	0.014	25369.000	257	98.712	148.000	89.000
1421	1548	pthread_mutex_lock	0.010	19021.000	184	103.375	175.000	90.000
1421	1548	pthread_mutex_unlock	0.010	18965.000	184	103.070	208.000	90.000

表 10-14 字段说明

字段名	字段解释
Name	对应调用API名称。
Type（仅AscendCL API）	调用AscendCL API的类型。
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。 可以单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。
Min(us)	对应API单次调用最短时间，单位为us。
Max(us)	对应API单次调用最长时间，单位为us。
Process ID	对应调用API所在进程ID。
Thread ID	对应调用API所在线程ID。

Runtime API

图 10-25 Runtime API

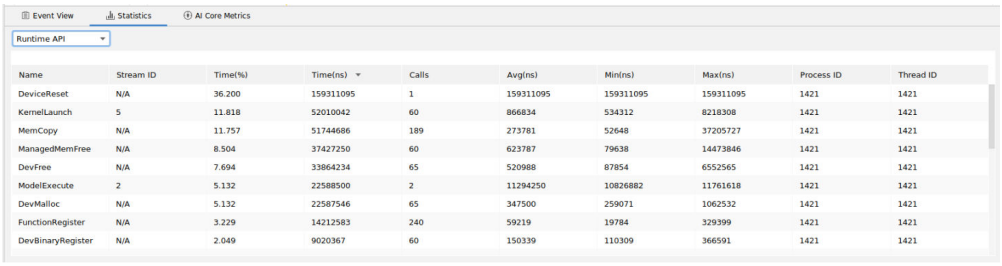


表 10-15 字段说明

字段名	字段解释
Name	对应调用API名称。
Stream ID	Stream ID，用于识别stream。
Time(%)	对应调用API总时间占比。
Time(ns)	对应调用API时间长度，单位为ns。 可以单击字段旁边的三角号根据此项进行降序或升序排列。
Calls	调用Runtime API的呼叫次数。
Avg(ns)	对应API单次调用平均时间，单位为ns。
Min(ns)	对应API单次调用最短时间，单位为ns。
Max(ns)	对应API单次调用最长时间，单位为ns。
Process ID	对应调用API所在进程ID。
Thread ID	对应调用API所在线程ID。

OPs

图 10-26 OPs（Atlas 200/300/500 推理产品和 Atlas 200/500 A2 推理产品）

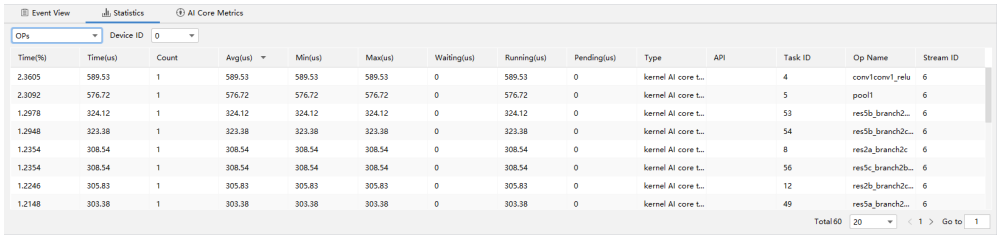


表 10-16 字段说明

字段名	字段解释
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。 可以单击字段旁边的三角号根据此项进行降序或升序排列。
Count	对应的Task被执行的次数。
Avg(us)	对应Task平均执行时间，单位为us。
Min(us)	对应Task最短执行时间，单位为us。
Max(us)	对应Task最长执行时间，单位为us。
Waiting(us)	对应Task总计waiting时间，单位为us。
Running(us)	对应Task总计running时间，单位为us。表示Task的实际运行时间，过长的Task运行时间可能意味着算子实现存在问题。
Pending(us)	对应Task总计pending时间，单位为us。
Type	对应Task的类型。
API	调用的接口。
Task ID	Task ID，用于识别task。
Op Name	算子名称。
Stream ID	Stream ID，用于识别stream。

图 10-27 OPs（Atlas 推理系列产品、Atlas A2 训练系列产品与 Atlas 训练系列产品）

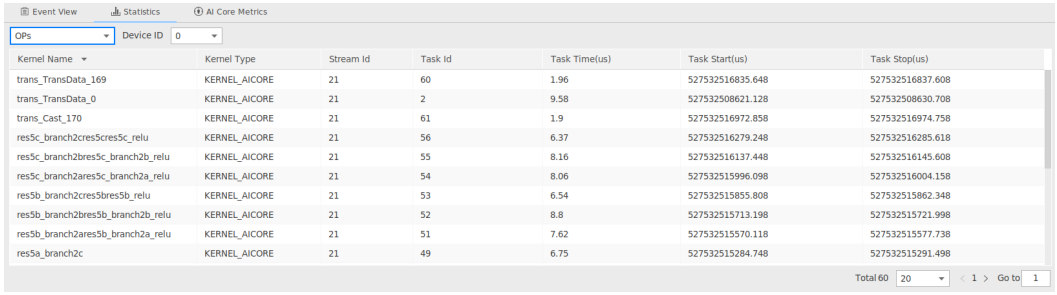


表 10-17 字段说明

字段名	字段解释
Kernel Name	kernel的名称。
Kernel Type	kernel的类型。

字段名	字段解释
Stream Id	Stream的ID号。
Task Id	Task的ID号。
Task Time(us)	Task总耗时，单位为us。
Task Start(us	Task开始时间，单位为us。
Task Stop(us)	Task结束时间，单位为us。

Op Info

图 10-28 Op Info

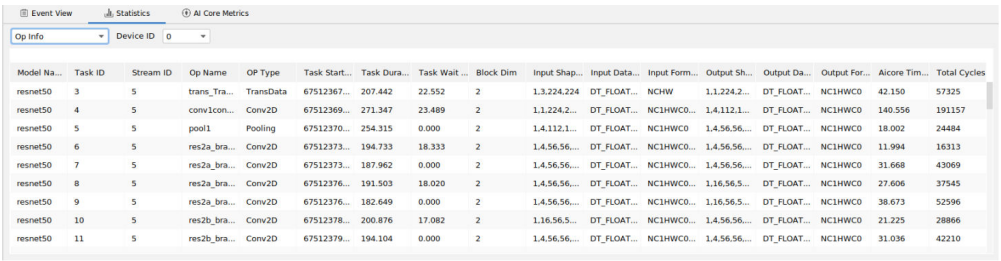


表 10-18 字段说明

字段名	字段解释
Model Name	模型名称。 推理应用中存在离线模型调用时（ACL函数接口：aclModelExecute），本参数展示；不存在离线模型调用时，本参数不展示。
Task ID	Task ID，用于识别task。
Stream ID	Stream ID，用于识别stream。
Op Name	算子名称。
Op Type	算子类型。
Task Start Time	Task的开始运行时间。
Task Duration(us)	Task的运行耗时，单位为us。
Task Wait Time(us)	上一个Task的结束时间与当前Task的开始时间间隔，单位为us。
Block Dim	Task运行时所在的核。

字段名	字段解释
Input Shapes	算子的输入维度。
Input Data Types	算子输入的数据类型。
Input Formats	算子输入格式。
Output Shapes	算子的输出维度。
Output Data Types	算子输出的数据类型。
Output Formats	算子输出格式。
Aicore Time(us)（仅采集方式为Task-Based时展示）	AI Core运行时间，单位为us。
Total Cycles（仅采集方式为Task-Based时展示）	该Task的所有指令的cycle总数。

10.8.2.5 AI Core Metrics 视图

单击下方数据窗格的AI Core Metrics，可以查看AI Core Metrics数据。

图 10-29 AI Core Metrics

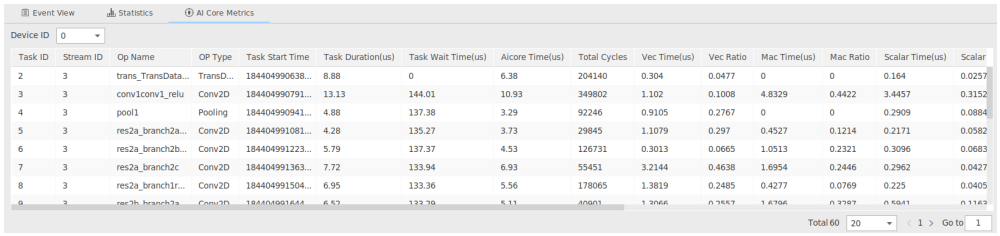


表 10-19 字段说明

字段名	字段解释
Task-based：Pipeline Utilization	
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。
OP Type	算子类型。
Task Start Time	任务启动时间。
Task Duration(us)	任务运行持续时间，单位为us。
Task Wait Time(us)	任务等待时间，单位为us。

字段名	字段解释
Aicore Time(us)	AI Core运行时间，单位为us。
Total Cycles	该Task的所有指令的cycle总数。
Vec Time(us)	vec类型指令（向量类运算指令）耗时，单位为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 Time(us)	cube类型指令（矩阵类运算指令）耗时，单位为us。
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 Time(us)	scalar类型指令（标量类运算指令）耗时，单位为us。
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 Time(us)	mte1类型指令（L1->L0A/L0B搬运类指令）耗时，单位为us。
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 Time(us)	mte2类型指令（DDR->AI Core搬运类指令）耗时，单位为us。
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 Time(us)	mte3类型指令（AI Core->DDR搬运类指令）耗时，单位为us。
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。
Task-based：Arithmetic Utilization	
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。
OP Type	算子类型。
Task Start Time	任务启动时间。
Task Duration(us)	任务运行持续时间，单位为us。

字段名	字段解释
Task Wait Time(us)	任务等待时间，单位为us。
Aicore Time(us)	AI Core运行时间，单位为us。
Total Cycles	该Task的所有指令的cycle总数。
Mac Fp16 Ratio	cube fp16类型指令的cycle数在所有指令的cycle数中的占用比。
Mac Int8 Ratio	cube int8类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Fp32 Ratio	vec fp32类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Fp16 Ratio	vec fp16类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Int32 Ratio	vec int32类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Misc Ratio	vec misc类型指令的cycle数在所有指令的cycle数中的占用比。
Cube Fops	cube类型的浮点运算数，即计算量，可用于衡量算法/模型的复杂度，其中Fops表示floating point operations，缩写为FLOPs。
Vector Fops	vector类型浮点运算数，即计算量，可用于衡量算法/模型的复杂度，其中Fops表示floating point operations，缩写为FLOPs。
Task-based：UB/L1/L2/Main Memory Bandwidth	
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。
OP Type	算子类型。
Task Start Time	任务启动时间。
Task Duration(us)	任务运行持续时间，单位为us。
Task Wait Time(us)	任务等待时间，单位为us。
Aicore Time(us)	AI Core运行时间，单位为us。
Total Cycles	该Task的所有指令的cycle总数。
ub_read_bw(GB/s)	Ub读带宽速率，单位为GB/s。
ub_write_bw(GB/s)	Ub写带宽速率，单位为GB/s。
l1_read_bw(GB/s)	L1读带宽速率，单位为GB/s。

字段名	字段解释
l1_write_bw(GB/s)	L1写带宽速率，单位为GB/s。
l2_read_bw(GB/s)	L2读带宽速率，单位为GB/s。
l2_write_bw(GB/s)	L2写带宽速率，单位为GB/s。
main_mem_read_bw(GB/s)	主存储器读带宽速率，单位为GB/s。
main_mem_write_bw(GB/s)	主存储器写带宽速率，单位为GB/s。
Task-based：L0A/L0B/L0C Memory Bandwidth	
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。
OP Type	算子类型。
Task Start Time	任务启动时间。
Task Duration(us)	任务运行持续时间，单位为us。
Task Wait Time(us)	任务等待时间，单位为us。
Aicore Time(us)	AI Core运行时间，单位为us。
Total Cycles	该Task的所有指令的cycle总数。
scalar_ld_ratio	scalar access ub类型读指令的cycle数在所有指令的cycle数中的占用比。
scalar_st_ratio	scalar access ub类型写指令的cycle数在所有指令的cycle数中的占用比。
l0a_read_bw(GB/s)	L0a读带宽速率，单位为GB/s。
l0a_write_bw(GB/s)	L0a写带宽速率，单位为GB/s。
l0b_read_bw(GB/s)	L0b读带宽速率，单位为GB/s。
l0b_write_bw(GB/s)	L0b写带宽速率，单位为GB/s。
l0c_read_bw(GB/s)	vector从l0c读带宽速率，单位为GB/s。
l0c_write_bw(GB/s)	vector向l0c写带宽速率，单位为GB/s。

字段名	字段解释
l0c_read_bw_cube(GB/s)	cube从l0c读带宽速率，单位为GB/s。
l0c_write_bw_cube(GB/s)	cube向l0c写带宽速率，单位GB/s。
Task-based：UB Memory Bandwidth	
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。
OP Type	算子类型。
Task Start Time	任务启动时间。
Task Duration(us)	任务运行持续时间，单位为us。
Task Wait Time(us)	任务等待时间，单位为us。
Aicore Time(us)	AI Core运行时间，单位为us。
Total Cycles	该Task的所有指令的cycle总数。
ub_read_bw_mte(GB/s)	mte从ub读带宽速率，单位为GB/s。Atlas 200/300/500 推理产品支持。
ub_write_bw_mte(GB/s)	mte向ub写带宽速率，单位为GB/s。Atlas 200/300/500 推理产品支持。
ub_read_bw_vector(GB/s)	vector从ub读带宽速率，单位为GB/s。
ub_write_bw_vector(GB/s)	vector向ub写带宽速率，单位为GB/s。
ub_read_bw_scalar(GB/s)	scalar从ub读带宽速率，单位为GB/s。
ub_write_bw_scalar(GB/s)	scalar向ub写带宽速率，单位为GB/s。
Sample-based：Pipeline Utilization	
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。
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。

字段名	字段解释
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。
Sample-based：Arithmetic Utilization	
Core ID	AI Core ID，用于识别AI Core。
Mac Fp16_ratio	cube fp16类型指令的cycle数在所有指令的cycle数中的占用比。
Mac Int8 Ratio	cube int8类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Fp32 Ratio	vec fp32类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Fp16 Ratio	vec fp16类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Int32 Ratio	vec int32类型指令的cycle数在所有指令的cycle数中的占用比。
Vec Misc Ratio	vec misc类型指令的cycle数在所有指令的cycle数中的占用比。
Cube Fops	Cube类型每秒浮点运算次数。
Vector Fops	Vector类型每秒浮点运算次数。
Sample-based：UB/L1/L2/Main Memory Bandwidth	
Core ID	AI Core ID，用于识别AI Core。
ub_read_bw(GB/s)	Ub读带宽速率，单位为GB/s。
ub_write_bw(GB/s)	Ub写带宽速率，单位为GB/s。
l1_read_bw(GB/s)	L1读带宽速率，单位为GB/s。
l1_write_bw(GB/s)	L1写带宽速率，单位为GB/s。
l2_read_bw(GB/s)	L2读带宽速率，单位为GB/s。
l2_write_bw(GB/s)	L2写带宽速率，单位为GB/s。

字段名	字段解释
main_mem_read_bw(GB/s)	主存储器读带宽速率，单位为GB/s。
main_mem_write_bw(GB/s)	主存储器写带宽速率，单位为GB/s。
Sample-based：L0A/L0B/L0C Memory Bandwidth	
Core ID	AI Core ID，用于识别AI Core。
l0a_read_bw(GB/s)	L0a读带宽速率，单位为GB/s。
l0a_write_bw(GB/s)	L0a写带宽速率，单位为GB/s。
l0b_read_bw(GB/s)	L0b读带宽速率，单位为GB/s。
l0b_write_bw(GB/s)	L0b写带宽速率，单位为GB/s。
l0c_read_bw(GB/s)	vector从l0c读带宽速率，单位为GB/s。
l0c_write_bw(GB/s)	vector向l0c写带宽速率，单位为GB/s。
l0c_read_bw_cube(GB/s)	cube从l0c读带宽速率，单位为GB/s。
l0c_write_bw_cube(GB/s)	cube向l0c写带宽速率，单位GB/s。
Sample-based：UB Memory Bandwidth	
Core ID	AI Core ID，用于识别AI Core。
ub_read_bw_vector(GB/s)	vector从ub读带宽速率，单位为GB/s。
ub_write_bw_vector(GB/s)	vector向ub写带宽速率，单位为GB/s。
ub_read_bw_scalar(GB/s)	scalar从ub读带宽速率，单位为GB/s。
ub_write_bw_scalar(GB/s)	scalar向ub写带宽速率，单位为GB/s。
ub_read_bw_mte(GB/s)	mte从ub读带宽速率，单位为GB/s。Atlas 200/300/500 推理产品支持。
ub_write_bw_mte(GB/s)	mte向ub写带宽速率，单位为GB/s。Atlas 200/300/500 推理产品支持。

10.8.3 Analysis Summary

Analysis Summary界面分为Analysis Summary和Collection And Platform Info两部分。

- Analysis Summary：通过**10.6 Import Result**导入已采集的集群场景PROF_XXX的父目录，可以查看并行性相关瓶颈的建议，如**图10-30**所示。
- Collection And Platform Info：通过选择需要查看的Device ID后，可以查看对应Device的详细硬件信息和Profiling采集信息，如**图10-31**所示。

图 10-30 Analysis Summary

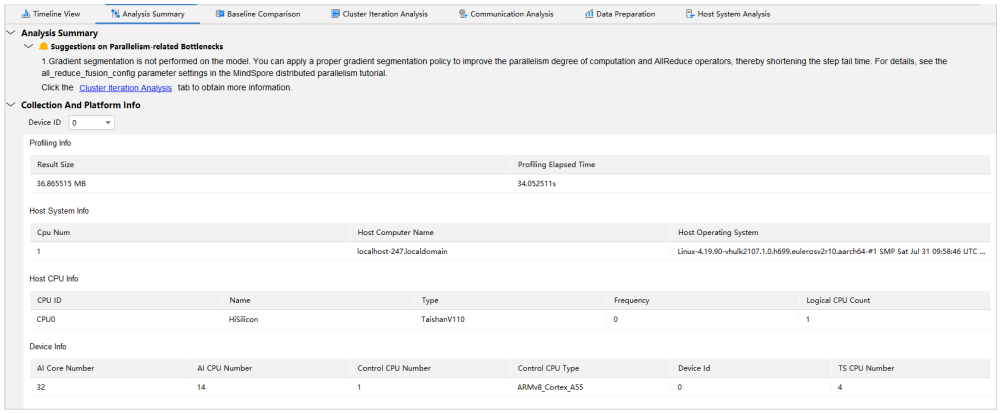
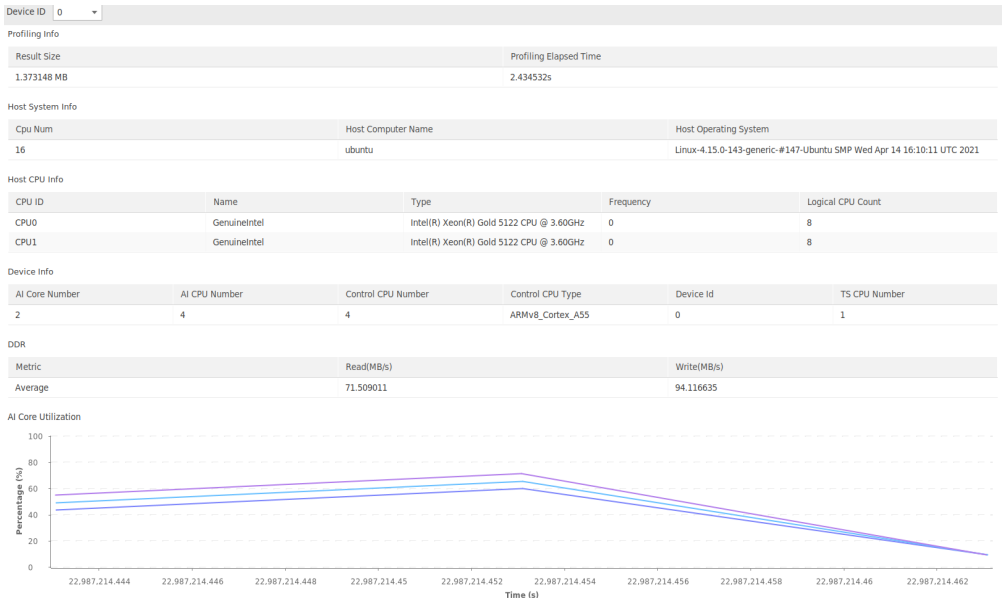


图 10-31 Collection And Platform Info



Analysis Summary

表 10-20 分析总结

字段	说明
Analysis Summary	分析汇总。

字段	说明
Suggestions on Parallelism-related Bottlenecks	关于并行性相关瓶颈的建议。
Click the Cluster Iteration Analysis tab to obtain more information.	单击Cluster Iteration Analysis获取更多信息。

Profiling Info

表 10-21 收集信息

字段	说明
Result Size	结果文件大小。
Profiling Elapsed Time	信息采集持续的时间。

Host System Info

表 10-22 Host 系统信息

字段	说明
Cpu Num	CPU数量。
Host Operating System	Host侧操作系统信息。
Host Computer Name	Host侧电脑名称。

Host CPU Info

表 10-23 Host 侧 CPU 信息

字段	说明
CPU ID	CPU ID。
Name	CPU名称。
Type	CPU型号。
Frequency	CPU频率。 部分系统由于不存在调用频率的接口，故不展示此参数，本参数展示情况请以实际情况为准。
Logical CPU Count	逻辑CPU数量。

Device Info

表 10-24 Device 信息

字段	说明
AI Core Number	AI Core数量。
AI CPU Number	AI CPU数量。
Control CPU Number	Control CPU数量。
Control CPU Type	Control CPU型号。
Device Id	当前页面关联的Device ID。
TS CPU Number	TS CPU数量。

DDR

表 10-25 DDR 参数说明

字段	说明
Metric	BandWidth，单位为MB/s。
Read(MB/s)	读带宽，单位为MB/s。
Write(MB/s)	写带宽，单位为MB/s。

AI Core Utilization

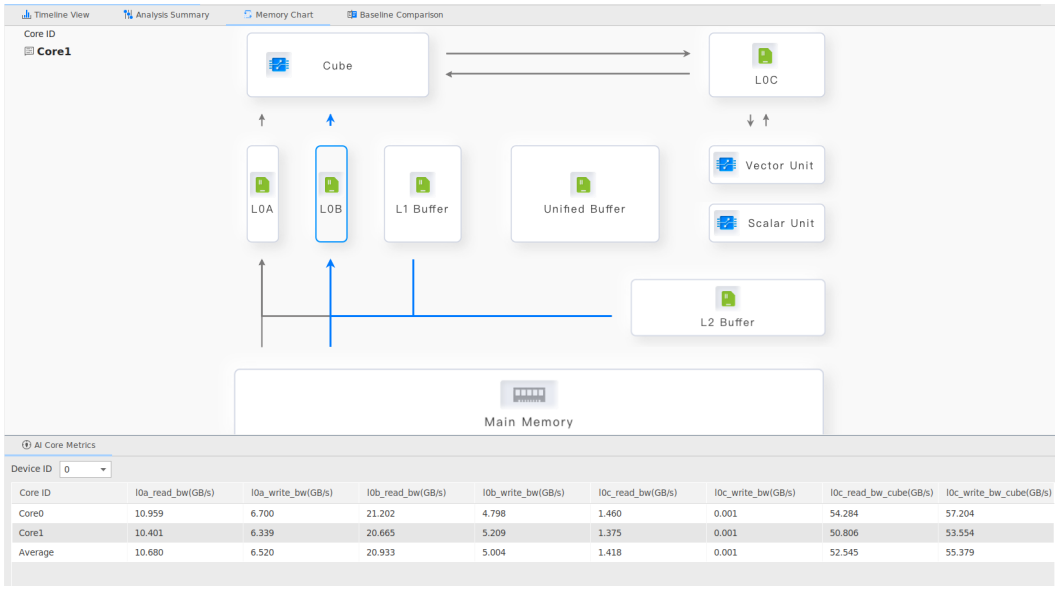
AI Core利用率通过折线图方式呈现（AI Core Utilization参数选择Sample-based才会展示）。

10.8.4 Memory Chart

Memory Chart视图通过将AI Core Metrics所展示的数据与对应硬件的组织结构图产生联动，使用户更加清晰理解Memory间关系及数据流向。

Memory Chart视图只展示计算单元和内存单元的读写速率和数据流向。如**图10-32**所示仅展示部分Memory Chart视图内容，完整字段解释请参见**表10-19**。

图 10-32 Memory Chart（仅为示例）



- 当AI Core采集开关为Pipeline Utilization和Arithmetic Utilization时，则无该视图展示。
- 当AI Core采集开关为UB/L1/L2/Main Memory Bandwidth时，单击UB/L1/L2/Main Memory单元则高亮显示对应单元及其数据流向箭头并且光标停靠显示该单元的采集数据。
- 当AI Core采集开关为L0A/L0B/L0C Memory Bandwidth时，单击L0A/L0B/L0C Memory单元则高亮显示对应单元及其数据流向并且光标停靠显示该单元的采集数据。
- 当AI Core采集开关为UB Memory Bandwidth时，单击Unified Buffer单元则高亮显示对应单元及其数据流向并且光标停靠显示该单元的采集数据。

10.8.5 Baseline Comparison

Profiling可以通过对比功能将两份报告的结果数据进行比对，并在Baseline Comparison（基准分析）视图展示。方便在重复执行Profiling产生多个报告后，对比相同情况下多次采集的数据之间的变化情况。

在打开的Profiling报告中选择Baseline Comparison视图如**图10-34**，单击Baseline file path右侧文件夹按钮，如**图10-34**选择需要与当前报告比对的基线数据报告（.json文件）。

图 10-33 Baseline Comparison（仅为示例）

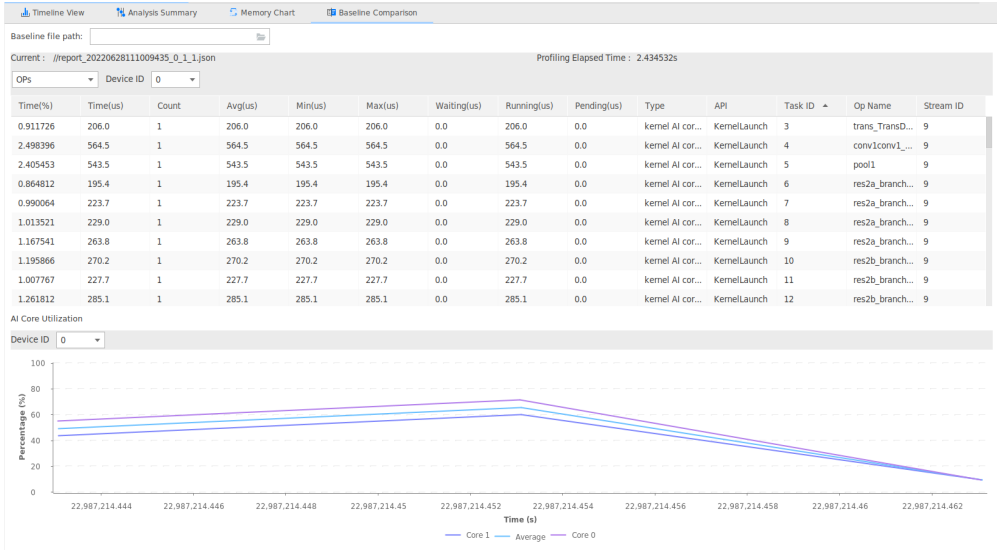
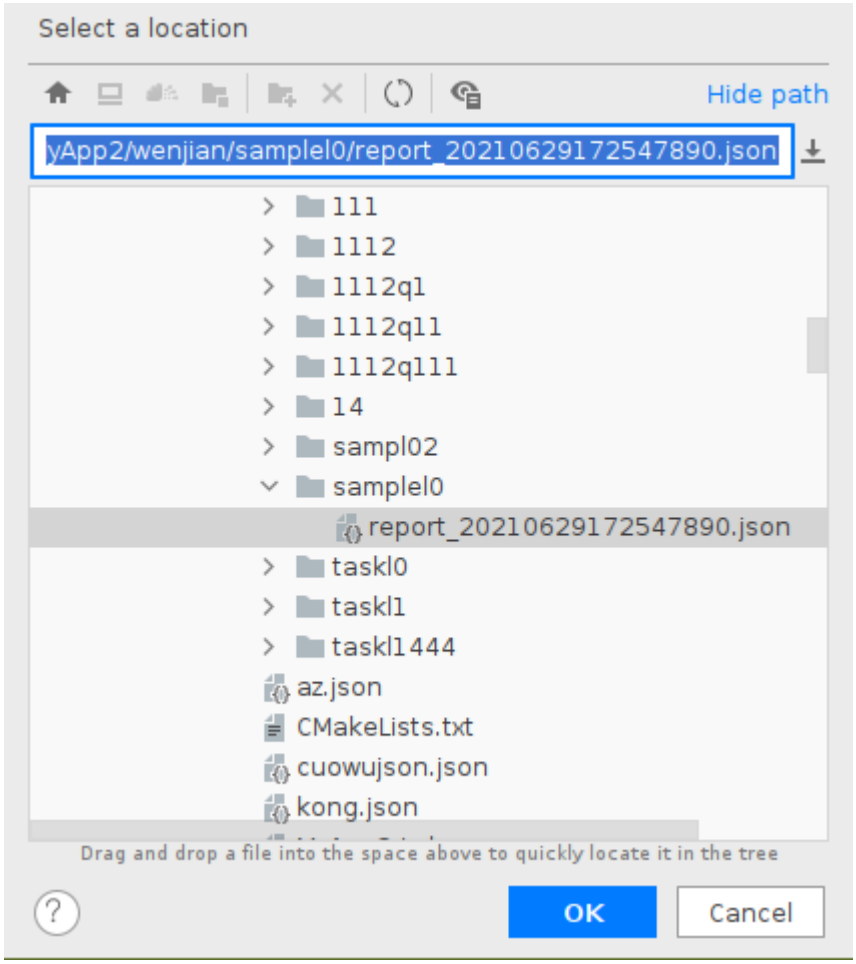


图 10-34 选择基线数据报告



两份报告比对结果如图10-35和图10-36，以Baseline file path所选报告的数据为基础，比对当前报告的数值以增减相应差值展示。

图 10-35 Task-based 场景比对结果

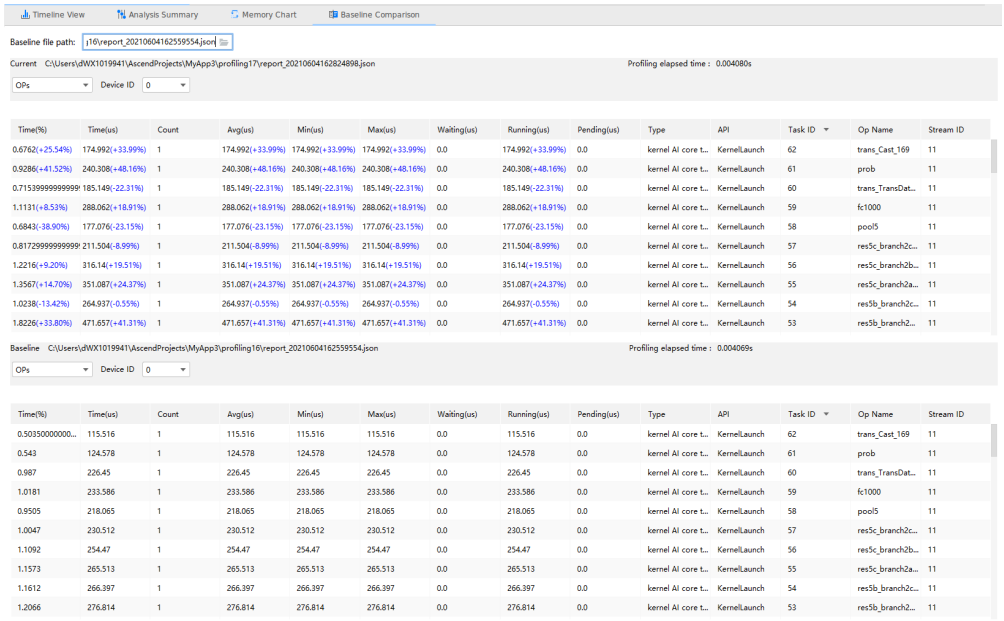
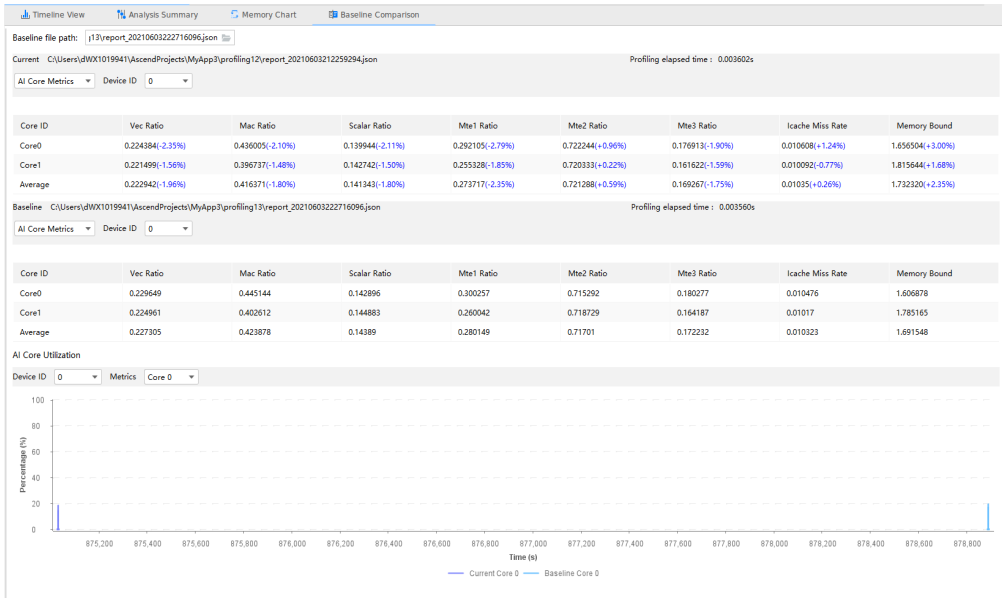


图 10-36 Sample-based 场景比对结果



说明

- Baseline Comparison仅比对OPs和AI Core Metrics的结果数据。
- Baseline Comparison仅支持相同Device ID的Profiling结果数据对比。
- 结果数据比对的差值保留小数点后两位。
- Profiling Elapsed Time表示本次Profiling执行采集的时间。
- 一般情况下两份报告须在相同设备、模型以及采集参数下Profiling生成的数据进行比对，否则结果数据不匹配，无法比较并给出相应告警提示；但如果两个不同设备或模型采集的数据完全相同，也可以进行比对。
- Sample-based场景展示AI Core Utilization的折线图比对结果。

10.8.6 集群场景数据

10.8.6.1 Cluster Iteration Analysis

Cluster Iteration Analysis训练集群场景迭代性能分析数据汇总，包含汇总页信息及每轮迭代的详细数据。

说明

MindStudio不支持集群场景的数据采集，可通过**10.6 Import Result**导入已采集的PROF_XXX的父目录来展示集群场景性能数据。

汇总页信息界面

首次进入Cluster Analysis页面时，展示汇总页信息，柱状图最多显示10组数据。

将汇总页信息界面分为**区域1和2**，详细字段解释请参见**表10-26**和**表10-27**。

当Type选择Iteration ID时，展示Step Trace和Data Parallelism Statistics/Model Parallelism Statistics/Pipeline Parallelism Statistics（三种并行模式仅展示其中一种），如**图10-37**/**图10-38**/**图10-39**所示。

当Type选择Rank ID时仅展示Step Trace，如**图10-40**所示。

图 10-37 Data Parallelism Statistics

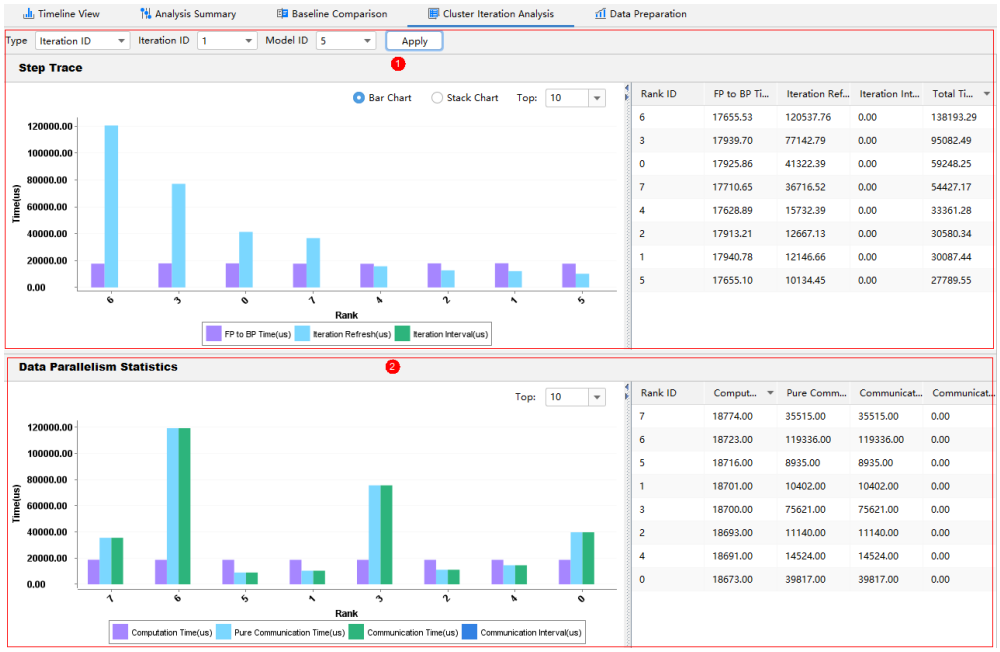


图 10-38 Model Parallelism Statistics

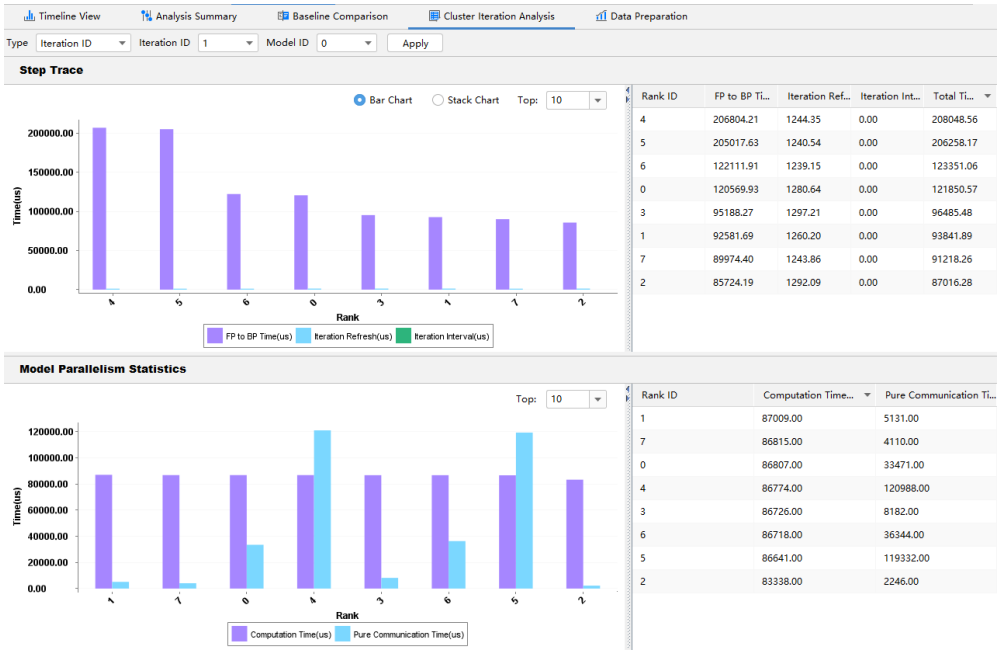


图 10-39 Pipeline Parallelism Statistics

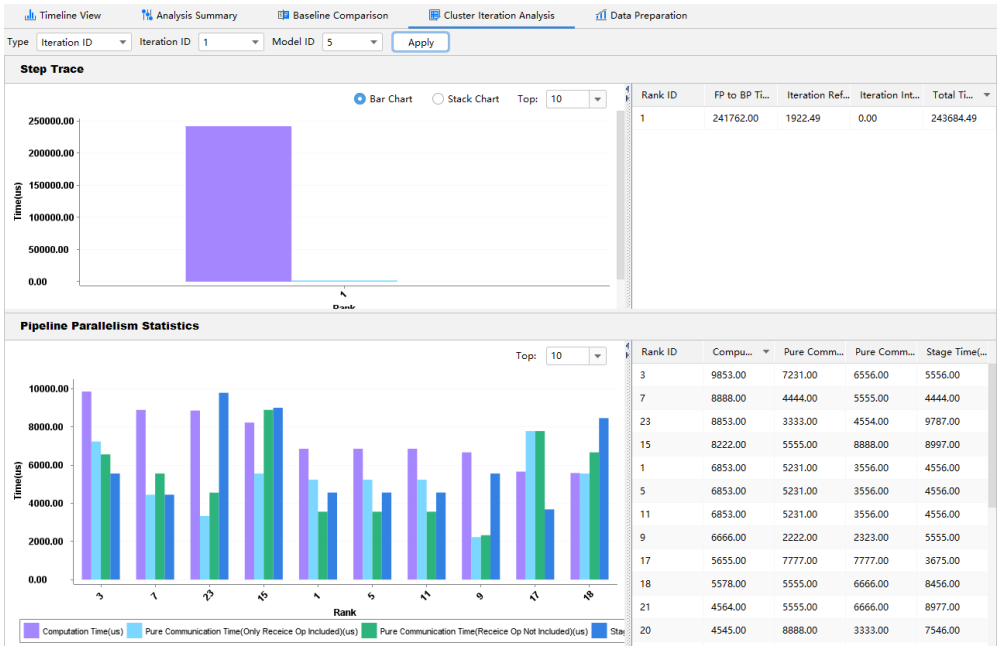
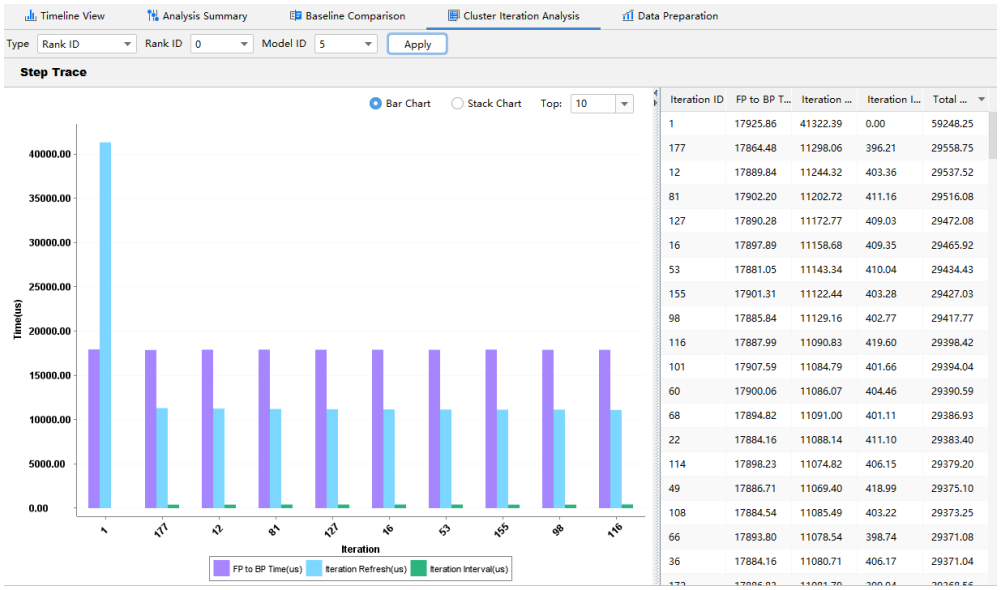


图 10-40 Rank ID



说明

- 单击汇总页信息界面中Step Trace某个柱状图时，弹出该Iteration ID/Rank ID的**迭代详细数据界面**。
- 柱状图横纵坐标说明如下：
  - 当Type选择Iteration ID时，横坐标表示从左至右根据所有集群节点的迭代轨迹默认按总耗时降序排列，并行度分析按计算时间降序排列，单击界面右侧表格上的列名时，则柱状图按照表格该列的数值排序，纵坐标表示耗时。
  - 当Type选择Rank ID时，横坐标表示从左至右根据当前集群节点所有的迭代轨迹默认按总耗时降序排列，单击界面右侧表格上的列名时，则柱状图按照表格该列的数值排序，纵坐标表示迭代耗时。

表 10-26 区域 1 字段说明

字段	说明
Type	数据展示方式： • Iteration ID（迭代ID）：Type选择Iteration ID并单击“Apply”时，下方柱状图显示当前迭代中所有集群节点的迭代数据。 • Rank ID（节点ID）：Type栏选择Rank ID并单击“Apply”时，下方柱状图显示当前集群节点所有迭代数据。
Iteration ID	迭代ID，查看指定迭代的所有设备迭代数据。
Rank ID	节点ID，查看指定节点的所有迭代数据。
Model ID	模型ID，查看指定迭代/节点的指定模型迭代数据。
Apply	数据导出按钮。当选定Iteration ID/Rank ID和Model ID并单击该按钮时，导出该节点的Cluster Iteration Analysis。
Step Trace（迭代轨迹数据）	

字段	说明
Bar Chart	柱状图展示迭代耗时数据。当选择此参数时，下方柱状图中的FP to BP Time、Iteration Refresh和Iteration Interval耗时数据以并排柱状图展示。
Stack Chart	堆叠图展示迭代耗时数据。当选择此参数时，下方柱状图中的FP to BP Time、Iteration Refresh和Iteration Interval耗时数据以堆叠柱状图展示。
Top	可通过配置Top参数值选择展示迭代总耗时最长的TopN条数据。取值范围1~200，默认值为10。
FP to BP Time(us)	FP/BP计算时间（BP End - FP Start）。单位为us。
Iteration Refresh(us)	迭代更新拖尾（Iteration End - BP End）。单位为us。
Iteration Interval(us)	迭代间隙。单位为us。
Total Time(us)	迭代总耗时。单位为us。

表 10-27 区域 2 字段说明

字段	说明
Rank ID	节点ID。
Top	可通过配置Top参数值选择展示集合通信总耗时最长的TopN条数据。取值范围1~200，默认值为10。
Data Parallelism Statistics（数据并行模式）	
Computation Time(us)	计算时间。单位为us。算子执行的时间总和，用于判断是否有慢卡存在。
Pure Communication Time(us)	纯通信时间。只有通信算子执行、计算算子不执行的时间段。单位为us。
Communication Time(us)	通信时间。单位为us。
Communication Interval(us)	通信间隙时间。单位为us。
Model Parallelism Statistics（模型并行模式）	
Computation Time(us)	计算时间。单位为us。算子执行的时间总和，用于判断是否有慢卡存在。
Pure Communication Time(us)	纯通信时间。单位为us。只有通信算子执行、计算算子不执行的时间段。
Pipeline Parallelism Statistics（流水线并行模式）	

字段	说明
Computation Time(us)	计算时间。单位为us。算子执行的时间总和，用于判断是否有慢卡存在。
Pure Communication Time (Only Receive Op Included) (us)	纯通信时间（仅包含Receive算子）。只有点对点（Receive）通信算子执行、计算算子不执行的时间段。单位为us。
Pure Communication Time (Receive Op Not Included) (us)	纯通信时间（不包含Receive算子）。只有除Receive通信算子外的其它通信算子执行、计算算子不执行的时间段。单位为us。
Stage Time(us)	Stage时间。单位为us。各个stage的耗时时长，查看该数据可以查看哪个stage的耗时最长。

迭代详细数据界面

单击汇总页信息界面中Step Trace某个柱状图时，弹出该Iteration ID/Rank ID的详细性能数据信息窗口，包括**区域1**（Timeline）、**区域2**（Bottleneck/Operator Statistics/Computing Workload）。如图10-41所示。

图 10-41 迭代详细数据界面

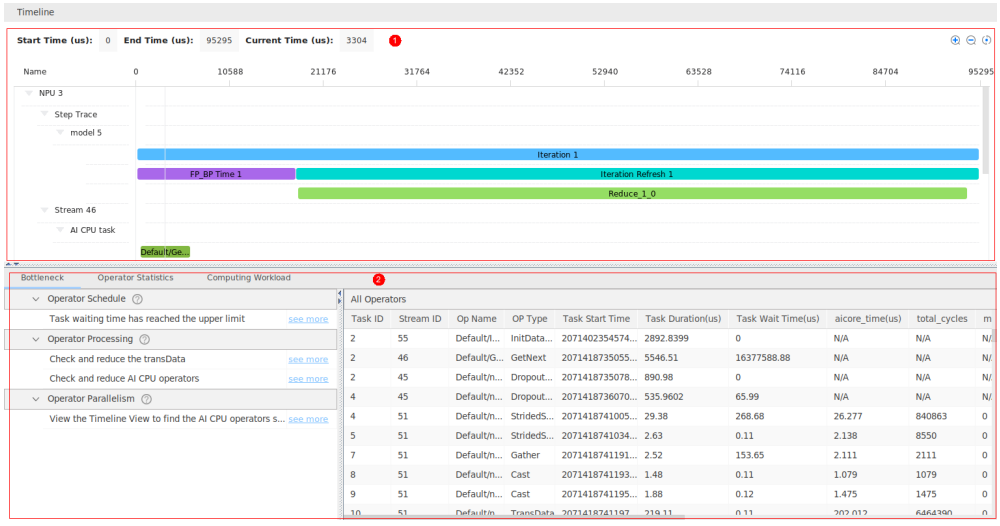


图 10-42 Operator Statistics

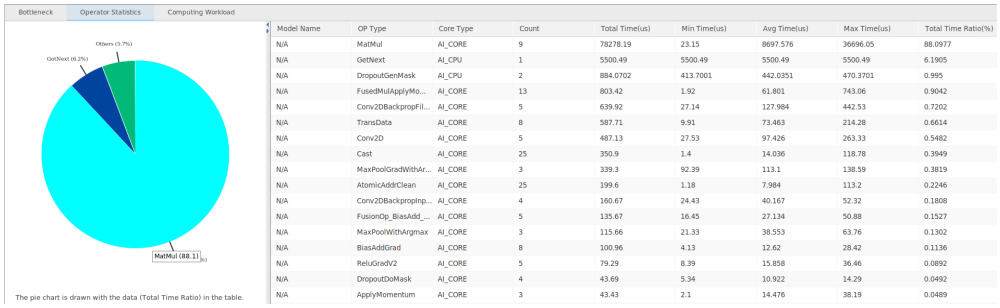
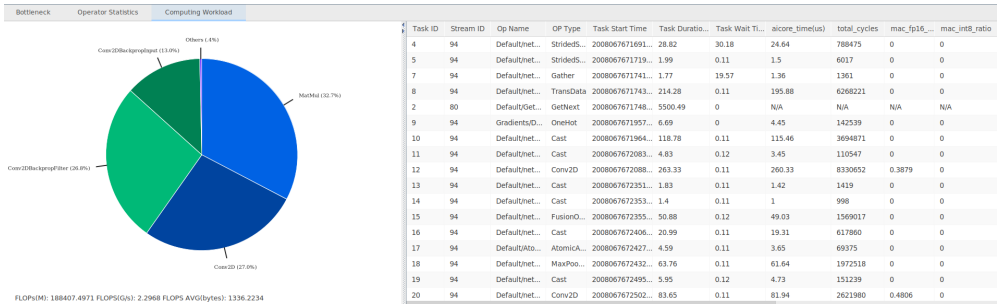


图 10-43 Computing Workload



区域1：

Timeline详细介绍请参见**10.8.2.2 Timeline视图**。

区域2：

- Bottleneck：瓶颈问题及优化建议。

  瓶颈问题分为六大类“Computation”、“Memory”、“Operator Schedule”、“Operator Processing”、“Operator Metrics”和“Operator Parallelism”，其中每类包含若干子问题，单击对应的“see more”后在右侧查看相关的算子信息，再次单击“see more”后查看全量的算子信息。

- Operator Statistics：算子统计，如**图10-42**所示。

  左侧饼图与右侧表格数据联动，单击某列表头时，饼图按照该列实际数据显示各数据所占比例。详细字段解释请参见**表10-28**。

  表 10-28 Operator Statistics 字段说明

字段	说明
Model Name	模型名称。如果Model Name值为空，则可能为获取的数据中该值为空。
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，单位us。
Min Time(us)	算子调用最小耗时，单位us。
Avg Time(us)	算子调用平均耗时，单位us。
Max Time(us)	算子调用最大耗时，单位us。
Total Time Ratio(%)	该类算子在对应模型中的耗时占比。

- Computing Workload：算子计算量，如**图10-43**所示 。

  饼图与右侧表格无联动，根据右侧表格中的OP Type列各算子类型所占比例绘制，且仅当性能数据采集为**Task-based**模式数据时才能展示。详细字段解释请参见**表10-29**。

表 10-29 Computing Workload 字段说明

字段	说明
FLOPs(M)	代表每秒所执行的浮点运算次数，FLOPs（floating-point operations per second）是计算机运算速度的单位。
FLOPS(G/s)	代表每秒浮点操作次数的峰值，FLOPS（floating-point operations per second, peak）是指计算机的峰值运算速度。
FLOPS AVG(bytes)	代表每秒浮点操作次数的平均值，FLOPS AVG（floating-point operations per second, average）是指计算机运算速度的平均值。
右侧表格中的字段展示与AI Core采集类型有关，各字段含义请参见**10.8.2.5 AI Core Metrics视图**。	

10.8.6.2 Data Preparation

Data Preparation数据准备性能分析。

说明

- 仅支持集群训练场景的**数据准备过程**进行性能分析。
- MindStudio不支持集群场景的数据采集，可通过**10.6 Import Result**导入已采集的PROF_XXX的父目录来展示集群场景性能数据。

数据准备过程可以分为三个阶段：数据处理pipeline、训练数据发送至Device以及Device侧读取训练数据。数据准备过程中，MindStudio性能分析工具通过迭代间隙的识别，实现了训练数据发送至Device以及Device侧读取训练数据两个阶段的性能瓶颈分析。

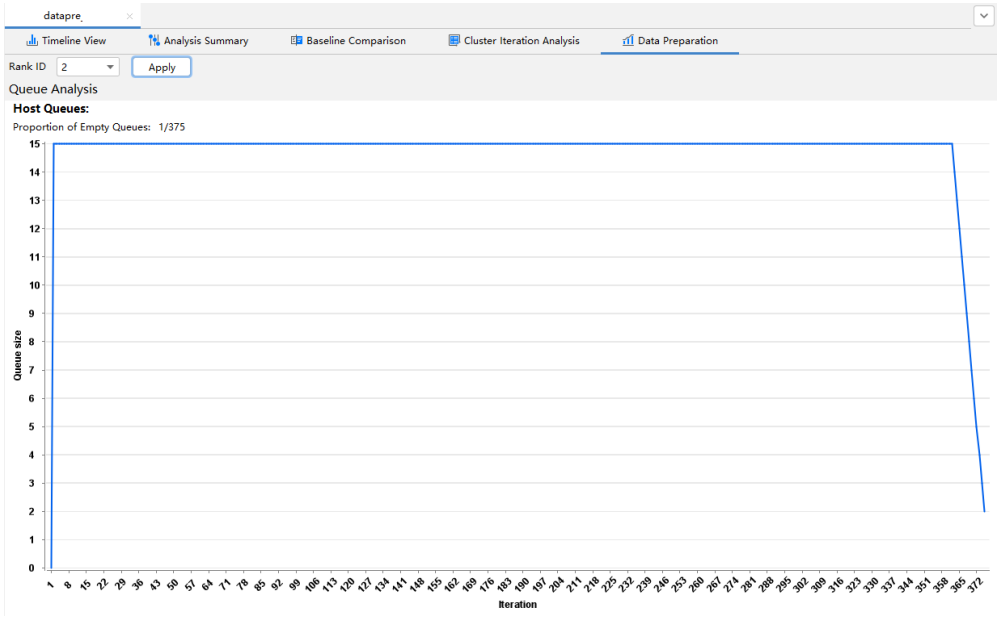
如**图10-44**所示为数据下沉模式，包含**Host Queues**（主机队列图）、**Data Queues**（数据队列图）、**Host Data Transmission**（主机数据发送图）和**Data Acquisition**（取数据算子耗时图）。

如**图10-45**所示为非数据下沉模式，只包含**Host Queues**（主机队列图）。

- **Data Queues**的纵坐标为Device侧读取训练数据时队列的长度。如果数据队列长度为0，则训练会一直等待，直到队列中有数据才会开始某个迭代的训练，该迭代可能存在性能瓶颈；如果数据队列长度大于0，则训练可以快速读取数据，数据准备不是该迭代的瓶颈所在；如果图中存在波动的曲线，则表示训练从队列读取数据时存在延迟，可能存在性能瓶颈。
- **Host Queues**的纵坐标为当前队列中缓存数据的个数。如果队列中缓存数据的个数在大部分情况下都是0，说明数据处理流程可能存在性能瓶颈点；如果队列中缓存数据的个数大于0，说明获取数据后将数据发送到Device侧的流程可能存在性能瓶颈。
- **Host Data Transmission**的纵坐标表示Host侧获取和推送数据的耗时，如果耗时较长，则可能存在性能瓶颈。
- **Data Acquisition**的纵坐标表示Device侧获取数据的耗时，如果耗时较长，则可能存在性能瓶颈。

图 10-44 数据下沉模式

图 10-45 非数据下沉模式



图中各字段说明如**表10-30**所示。

表 10-30 字段说明

字段	说明
Rank ID	集群场景的节点ID。

字段	说明
Apply	数据导出按钮。当选择某个Rank ID并单击该按钮时，导出该节点的Data Preparation。
Queues Analysis	
Host Queues	主机队列图。
Data Queues	数据队列图。
Proportion of Empty Queues: */*	队列为空比例：空队列数/总队列数。为主机和数据队列图横纵坐标信息的汇总值。
Iteration	迭代。
Queue size	队列个数。
Consumption Analysis	
Host Data Transmission	主机数据发送图。
Average Duration: *ms	平均总耗时，单位为ms。取主机数据发送图横纵坐标信息的汇总后的平均值。
Average Data Acquisition Duration: *ms	平均取数据耗时，单位为ms。
Average Data Sending Duration: *ms	平均推送数据耗时，单位为ms。
Total duration	总耗时。取主机数据发送图横纵坐标信息的汇总。
Data acquisition duration	取数据耗时。
Data sending duration	推送数据耗时。
Data Acquisition	取数据算子耗时图。
Average Duration: *ms	平均总耗时，单位为ms。为取数据算子耗时图横纵坐标信息的汇总后的平均值。
Iteration	迭代。
Time(ms)	耗时。

10.8.6.3 Communication Analysis

Communication Analysis通信性能分析，展示了集群场景中所有卡的通信性能以及全网链路性能。

说明

MindStudio不支持集群场景的数据采集，可通过**10.6 Import Result**导入已采集的PROF_XXX的父目录来展示集群场景性能数据。

Communication Analysis界面分为Communication Duration Analysis和Communication Matrix两部分进行数据展示。

- Communication Duration Analysis：通信耗时分析，主要展示卡的通信性能，包括通信时长、等待时长以及卡的链路信息等，如图10-46所示。
- Communication Matrix：通信矩阵，主要展示三种链路方式的相关信息，包括带宽、通信时长及通信尺寸等，如图10-47所示。

图 10-46 Communication Duration Analysis

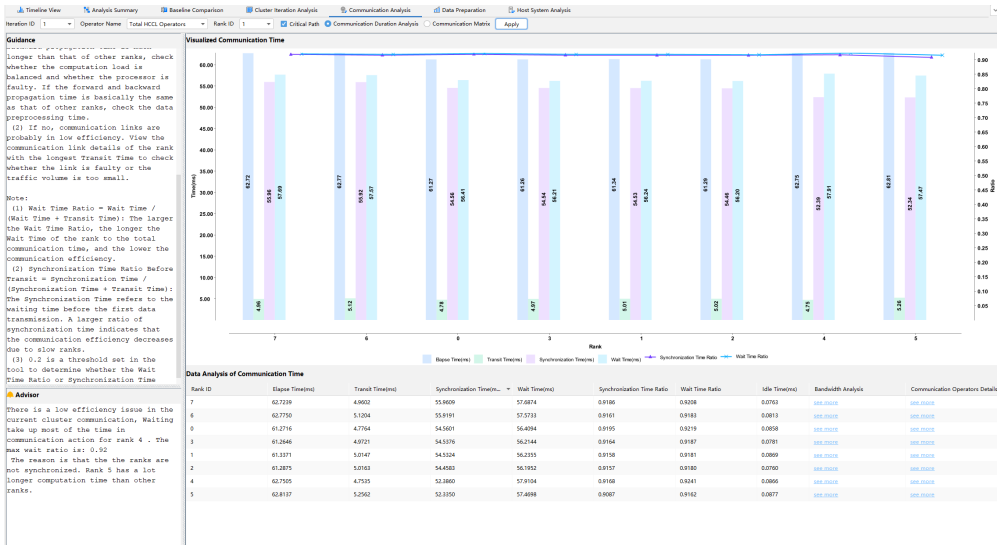
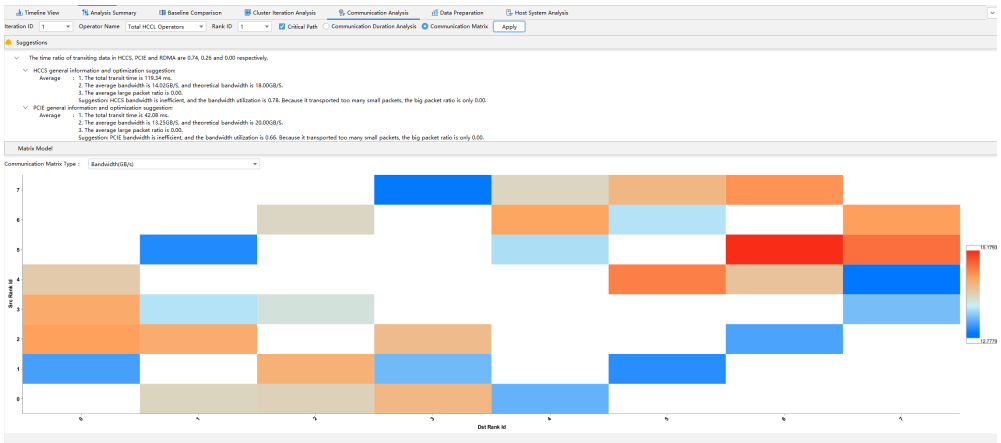


图 10-47 Communication Matrix



图中各字段说明如表10-31所示。

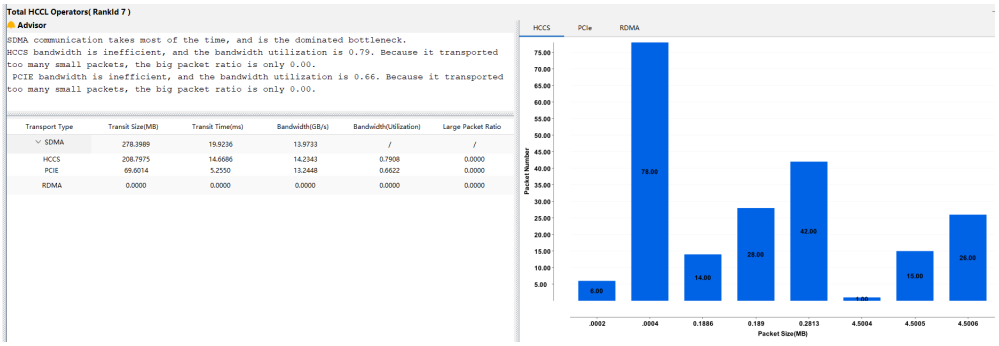
表 10-31 字段说明

字段	说明
Iteration ID	迭代ID，查看指定迭代的所有算子迭代数据。
Operator Name	通信算子名称，查看指定算子的迭代数据。

字段	说明
Rank ID	节点ID，查看指定节点的所有算子迭代数据。 ● 启用“Critical Path”时，Rank ID的下拉框支持选择。 ● 关闭“Critical Path”时，Rank ID的下拉框置灰，不支持选择。
Critical Path	关键路径，筛选出关键Task路径下的性能数据，用于分析和展示。默认启用。
Apply	数据导出按钮。 ● 先选择Communication Duration Analysis，再选择某个Iteration ID/Operator Name并单击该按钮，导出该迭代下指定算子的通信耗时分析数据。 ● 先选择Communication Matrix，再选择某个Iteration ID/Operator Name并单击该按钮，导出该迭代下指定算子的通信矩阵分析数据。
Communication Duration Analysis	通信耗时分析。
Guidance	指引。可根据指引信息进行查看，如检查Rank的等待时长比例是否大于阈值（0.2）。
Advisor	慢卡/慢节点分析建议。
Visualized Communication Time	可视化通信时长。
Time(ms)	时长。
Rank	节点。集群场景的某个节点。
Ratio	比例。包含Synchronization Time Ratio和Wait Time Ratio。
Data Analysis of Communication Time	算子的通信时长数据分析。
Rank ID	集群场景的节点ID。
Elapse Time(ms)	算子的通信总耗时。
Transit Time(ms)	通信时长。表示通信算子的通信耗时，如果通信耗时过长，可能是某条链路存在问题。
Synchronization Time(ms)	同步时长。卡之间进行同步需要的时长。
Wait Time(ms)	等待时长。卡之间进行通信前，首先会进行同步，确保通信的两张卡同步完成，再进行通信。

字段	说明
Synchronization Time Ratio	同步时长比例。 同步时长比例（Synchronization Time Ratio）= 同步时长（Synchronization Time）/ (同步时长（Synchronization Time）+ 通信时长（Transit Time）)，通信前的同步时长比例越大说明通信效率越低，可能存在慢卡的情况。
Wait Time Ratio	通信算子的等待时长比例。 等待时长比例（Wait Time Ratio）= 等待时长（Wait Time）/ (等待时长（Wait Time）+ 通信时长（Transit Time）)，等待时长比例越大代表卡的等待时长占总通信耗时越长，通信效率越低。
Idle Time(ms)	通信算子下发耗时。 通信算子下发耗时（Idle Time）= 算子的通信总耗时（Elapse Time）- 通信时长（Transit Time）- 等待时长（Wait Time）。
Bandwidth Analysis	带宽分析。单击对应的“see more”后可查看指定算子的带宽详情，如图10-48所示。
Communication Operators Details	通信算子的详情。单击对应的“see more”后可查看通信算子的链路详情，如图10-49所示。
Communication Matrix	通信矩阵。
Suggestions	分析建议。基于不同链路方式（HCCS、PCIe和RDMA）对全网链路信息给出分析建议，包含通信时长、通信带宽、通信量、通信带宽使用率以及慢链路等。
Matrix Model	矩阵模型。
Communication Matrix Type	通信矩阵类型。 • Bandwidth(GB/s)：带宽。 • Transit Size(MB)：通信尺寸。 • Transport Type：链路类型。 • Large Packet Ratio：大通信包比例。 • Bandwidth(Utilization)：带宽使用率。 • Transit Time(ms)：通信时长。
Src Rank Id	Source Rank Id，逻辑卡链路信息中源卡的节点Id。
Dst Rank Id	Destination Rank Id，逻辑卡链路信息中目的卡的节点Id。

图 10-48 Bandwidth Analysis

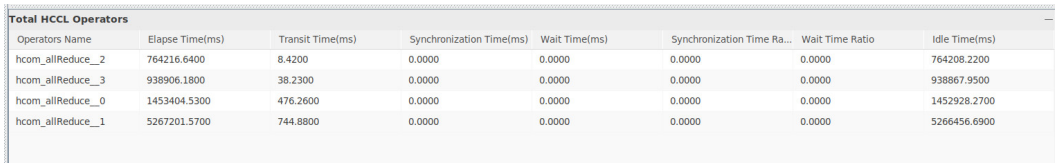


图中各字段说明如表10-32所示。

表 10-32 字段说明

字段	说明
Advisor	分析建议。
Transport Type	链路方式。
SDMA	SDMA链路。包括HCCS和PCIE两种。
HCCS	HCCS链路。
PCIE	PCIE链路。
RDMA	RDMA链路。
Packet Number	通信包数量。
Packet Size(MB)	通信包大小。
Transit Size(MB)	一次通信包的大小。
Transit Time(ms)	一次通信的时长。
Bandwidth(GB/s)	带宽。带宽一般为通信量除以通信时间。
Bandwidth(Utilization)	带宽使用率。如果实际带宽小于0.8倍的经验带宽，则说明带宽使用率不高，需进一步分析。 经验带宽参考值分别为RDMA_Bandwidth = 12.5，HCCS_Bandwidth = 18， PCIe_Bandwidth = 20。
Large Packet Ratio	大通信包比例。通信包的大小足以使得通信链路能达到经验带宽的包的比率。

图 10-49 Communication Operators Details



图中各字段说明如表10-33所示。

表 10-33 字段说明

字段	说明
Operator Name	通信算子名称。
Elapse Time(ms)	通信算子所有事件消耗时间之和，单位ms。
Transit Time(ms)	通信时长，单位ms。通信时长的计算方式为统计SDMA链路和RDMA链路的通信算子总耗时。
Synchronization Time(ms)	同步时长，单位ms。第一次传输数据前的等待时间。
Wait Time(ms)	等待时长，单位ms。逻辑卡之间进行通信前，首先会进行同步，确保通信的两张卡同步完成，再进行通信。
Synchronization Time Ratio	同步时长比例。计算公式为Synchronization Time / (Synchronization Time + Transit Time)。
Wait Time Ratio	等待时长比例。计算公式为Wait Time / (Wait Time + Transit Time)。
Idle Time(ms)	通信算子下发耗时。 通信算子下发耗时（Idle Time）= 算子的通信总耗时（Elapse Time）- 通信时长（Transit Time）- 等待时长（Wait Time）。

10.8.7 Host System Analysis

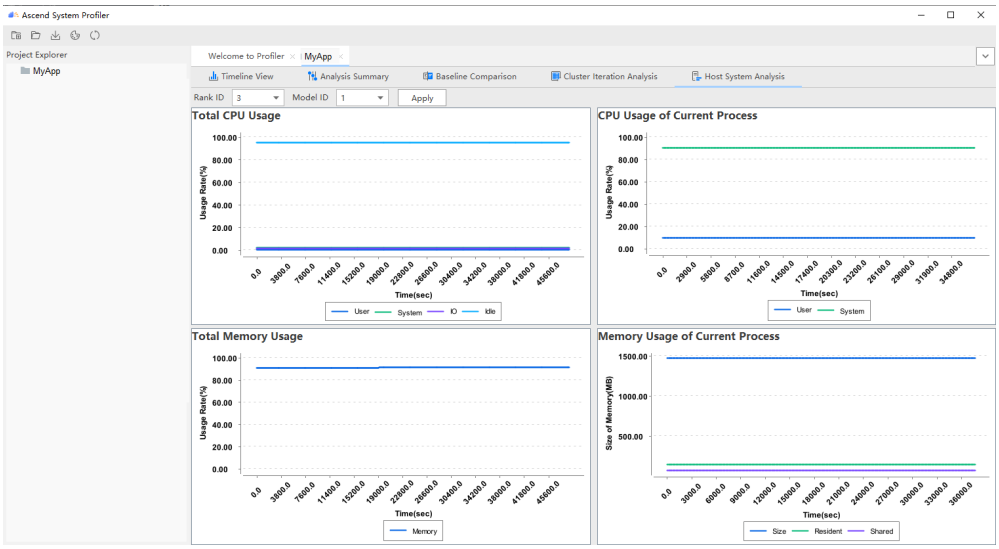
Host System Analysis界面分为Total CPU Usage、CPU Usage of Current Process、Total Memory Usage和Memory Usage of Current Process四部分，如图10-50所示。

- Total CPU Usage：展示Host侧系统整体的CPU使用情况，包含用户利用率、系统利用率、空闲利用率和IO利用率。
- CPU Usage of Current Process：展示Host侧进程的CPU使用情况，包含用户利用率和系统利用率。
- Total Memory Usage：展示Host侧系统整体的内存利用率。
- Memory Usage of Current Process：展示Host侧进程的内存使用情况，包含虚拟地址空间、物理内存和共享内存的大小。

说明

在任一部分选中某个时间点后，可以通过向右拖拉鼠标再释放实现视图的放大，以便查看细节，双击鼠标左键后视图还原。

图 10-50 Host System Analysis



Total CPU Usage

表 10-34 Total CPU Usage

字段	说明
Total CPU Usage	系统CPU利用率。
Usage Rate(%)	利用率。
Time(sec)	采集间隔时间。
User	用户利用率。如果用户利用率较低，可以尝试增大线程数，增加CPU使用情况。
System	系统利用率。如果系统利用率较大，CPU等待处理的进程较大，说明需要相应减少线程个数。
IO	IO利用率。
Idle	空闲利用率。

CPU Usage of Current Process

表 10-35 CPU Usage of Current Process

字段	说明
CPU Usage of Current Process	进程CPU利用率。展示单个进程的CPU占用情况。
Usage Rate(%)	利用率。

字段	说明
Time(sec)	采集间隔时间。
User	用户利用率。
System	系统利用率。

Total Memory Usage

表 10-36 Total Memory Usage

字段	说明
Total Memory Usage	系统内存利用率。
Usage Rate(%)	利用率。
Time(sec)	采集间隔时间。
Memory	系统内存。

Memory Usage of Current Process

表 10-37 Memory Usage of Current Process

字段	说明
Memory Usage of Current Process	进程内存利用率。
Size of Memory(MB)	内存大小。
Time(sec)	采集间隔时间。
Size	进程占用内存大小。
Resident	进程占用的物理内存大小。
Shared	进程占用的共享内存大小。

10.9 性能分析样例参考

10.9.1 网络应用中的函数计算性能优化分析样例

背景介绍

使用PyTorch网络应用在昇腾平台执行推理过程中，发现整体执行时间较长。为了找出原因，使用性能分析工具对该网络应用执行推理耗时分析，分析结果显示运行的接口aclmdlExecute执行耗时数值较高，进一步分析发现Conv算子执行时间最长。

打开PyTorch网络转换成的om模型查询Conv算子，发现该算子由多个计算单元组成，会造成极大的推理开销。由于Conv算子所在函数为Mish激活函数，而当前昇腾平台支持的激活函数只有：Relu、Leakyrelu、Prelu、Elu、Srelu，Mish函数暂时不支持，因此造成模型转换后的Mish函数被分解成了多个计算单元。

通过将om模型中的Mish函数替换昇腾平台的激活函数，尝试降低推理耗时，以Leakyrelu替换Mish函数为例，重新执行Profiling性能分析，结果发现推理耗时明显降低。

说明

- 本文仅介绍Profiling性能分析工具的操作及分析过程，对于om模型的算子分析以及函数替换等操作此处不做阐述。
- 有关PyTorch模型转换为om模型的详细介绍请参见《**PyTorch网络迁移和训练指南**》中的“**保存与导出模型**”章节。

Profiling 性能分析操作

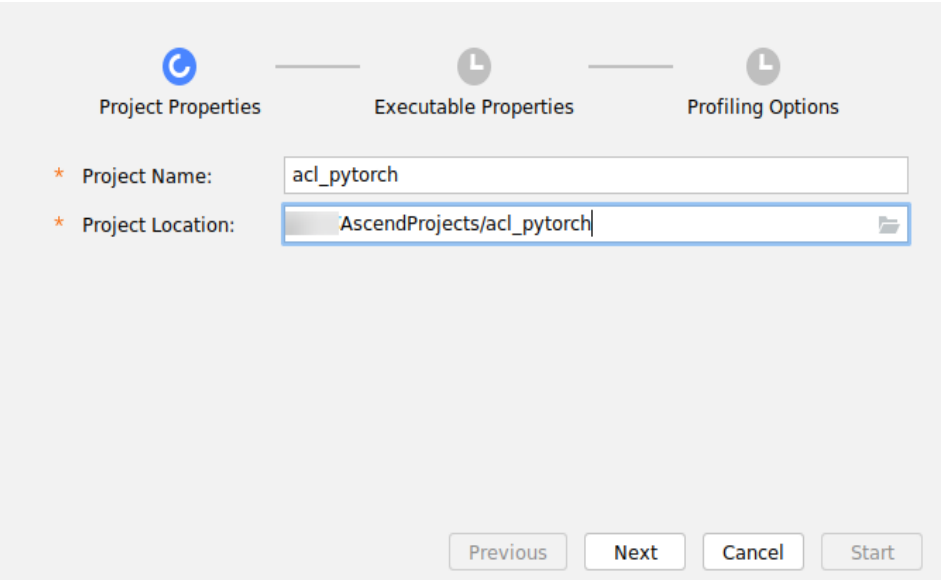
步骤1 启动MindStudio，单击选择并打开已编译完成的工程。

有关应用工程的创建和编译操作请参见**7 应用开发**。

步骤2 单击菜单栏“Ascend > System Profiler > New Project”，弹出Profiling配置窗口 。

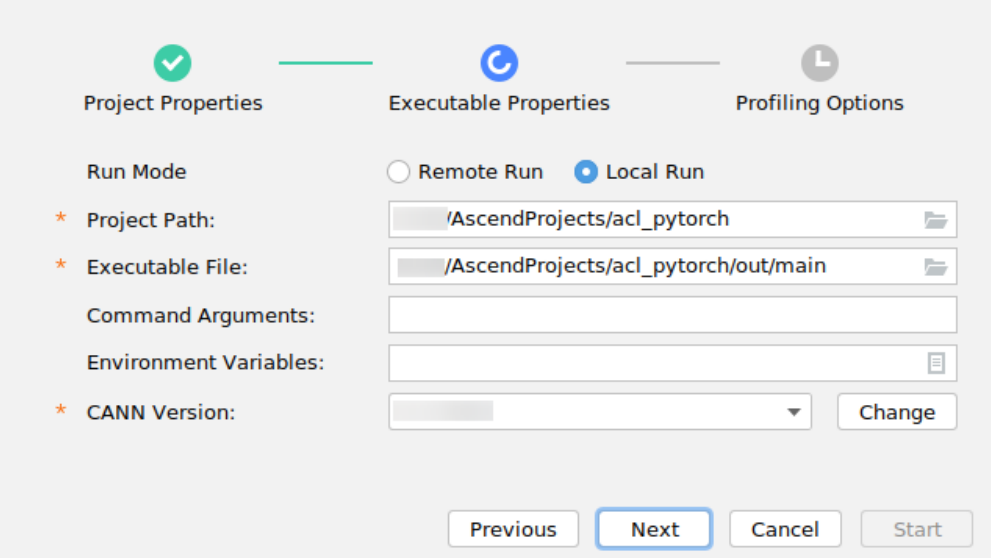
步骤3 进入Profiling配置窗口，如**图10-51**。配置Profiling的工程名称“Project Name”和选择Profiling工程的结果路径“Project Location”。单击“Next”进入下一步。

图 10-51 Project Properties 配置



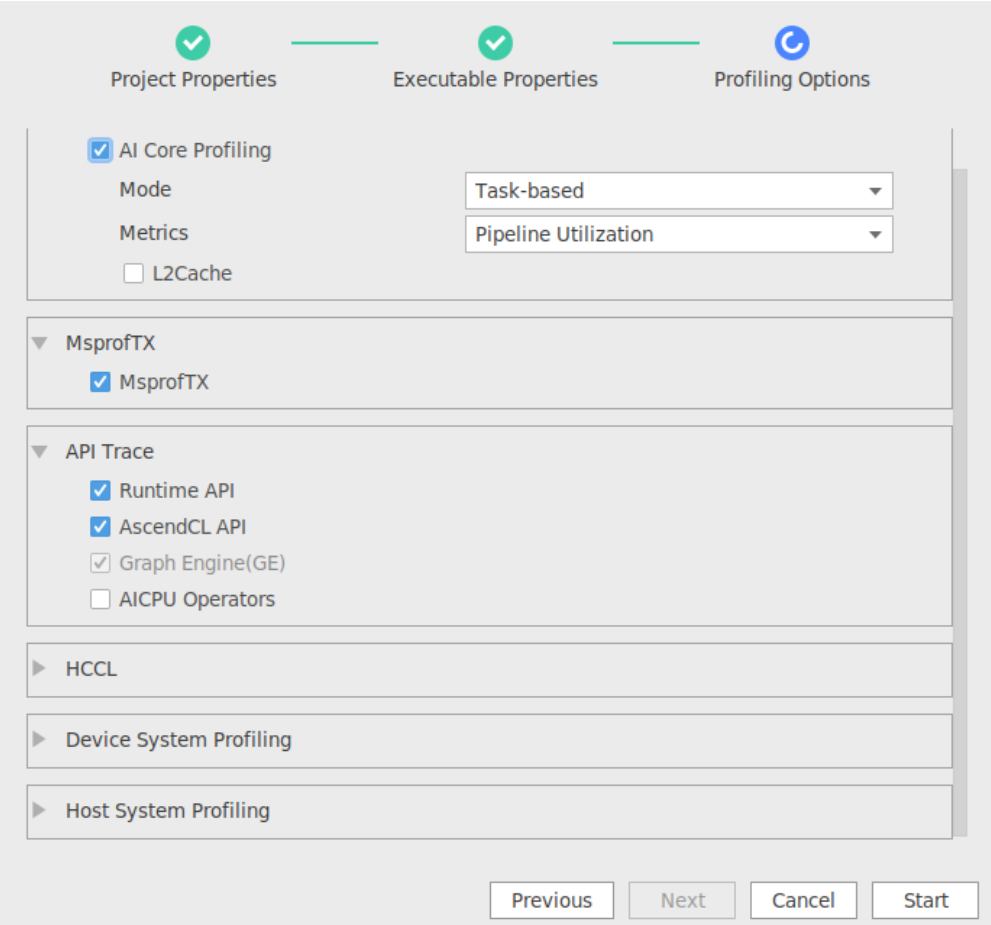
步骤4 进入“Executable Properties”配置界面，选择Local Run模式。指定执行Profiling目标工程的可执行文件目录。如图10-52所示。

图 10-52 Local Run



步骤5 进入“Profiling Options”配置界面，选择Task-based场景。如图10-53所示。

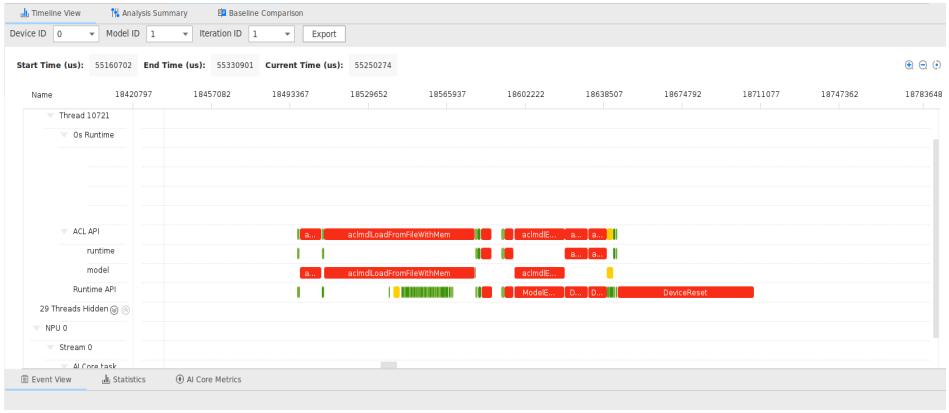
图 10-53 Task-based 场景



步骤6 完成上述配置后单击窗口右下角的“Start”按钮，启动Profiling。

工程执行完成后，MindStudio自动展示Profiling结果视图。如图10-54所示。

图 10-54 Profiling 分析结果



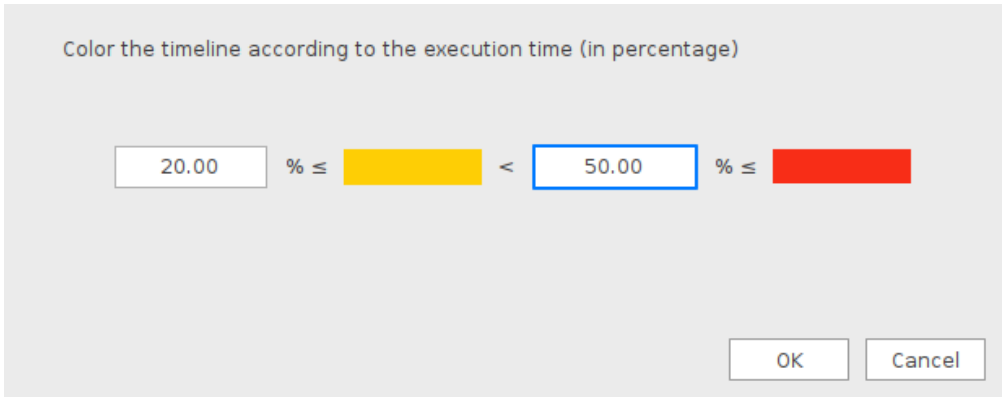
----结束

问题分析

步骤1 根据Profiling **Timeline视图**的字段解释，展示模型、算子的耗时数据体现在AscendCL API字段。

通过设置**10.8.2.1 Timeline颜色配置**调色板，设置显色比例为20%<=黄色<50%<=红色，如图10-55所示。

图 10-55 Import Result 设置

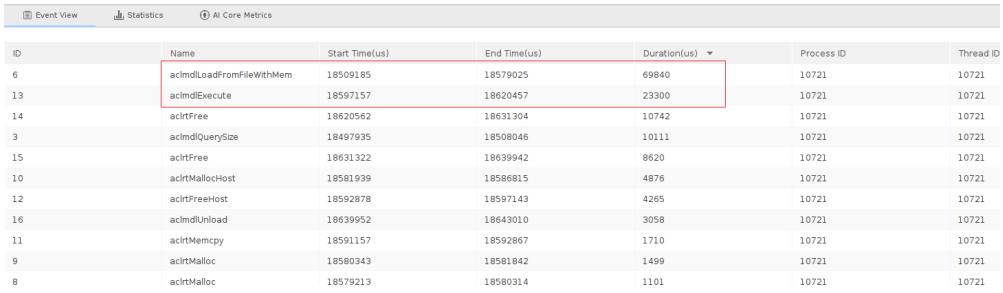


得到新的Timeline比例并放大视图，如图10-56所示。此时可以直观的看到耗时最长的时间线有两段，分别为aclmdlLoadFromFileWithMem和aclmdlExcute接口。

图 10-56 AscendCL API Timeline

步骤2 接着在AscendCL API字段上右键单击“Show in Event View”并将Duration列按从大到小排序，如图10-57所示。

图 10-57 Event View

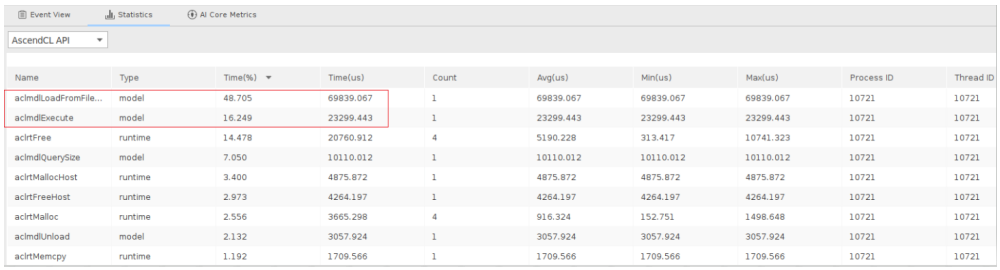


ID	Name	Start Time(us)	End Time(us)	Duration(us)	Process ID	Thread ID
6	aclmdlLoadFromFileWithMem	18509185	18579025	69840	10721	10721
13	aclmdlExecute	18597157	18620457	23300	10721	10721
14	aclrtFree	18620562	18631304	10742	10721	10721
3	aclmdlQuerySize	18497935	18508046	10111	10721	10721
15	aclrtFree	18631322	18639942	8620	10721	10721
10	aclrtMallocHost	18581939	18586815	4876	10721	10721
12	aclrtFreeHost	18592878	18597143	4265	10721	10721
16	aclmdlUnload	18639952	18643010	3058	10721	10721
11	aclrtMemcpy	18591157	18592867	1710	10721	10721
9	aclrtMalloc	18580343	18581842	1499	10721	10721
8	aclrtMalloc	18579213	18580314	1101	10721	10721

步骤3 可以看到耗时最高的两个AscendCL接口为aclmdlLoadFromFileWithMem和aclmdlExcute。

继续查看AscendCL API的**Statistics视图**并按接口调用耗时占比从高到底排序，如图10-58所示。

图 10-58 AscendCL API Statistics



Name	Type	Time(%)	Time(us)	Count	Avg(us)	Min(us)	Max(us)	Process ID	Thread ID
aclmdlLoadFromFile...	model	48.705	69839.067	1	69839.067	69839.067	69839.067	10721	10721
aclmdlExecute	model	16.249	23299.443	1	23299.443	23299.443	23299.443	10721	10721
aclrtFree	runtime	14.478	20760.912	4	5190.228	313.417	10741.323	10721	10721
aclmdlQuerySize	model	7.050	10110.012	1	10110.012	10110.012	10110.012	10721	10721
aclrtMallocHost	runtime	3.400	4875.872	1	4875.872	4875.872	4875.872	10721	10721
aclrtFreeHost	runtime	2.973	4264.197	1	4264.197	4264.197	4264.197	10721	10721
aclrtMalloc	runtime	2.556	3665.298	4	916.324	152.751	1498.648	10721	10721
aclmdlUnload	model	2.132	3057.924	1	3057.924	3057.924	3057.924	10721	10721
aclrtMemcpy	runtime	1.192	1709.566	1	1709.566	1709.566	1709.566	10721	10721

到这里基本可以断定执行应用推理过程中耗时最高的两个接口就是aclmdlLoadFromFileWithMem和aclmdlExcute。

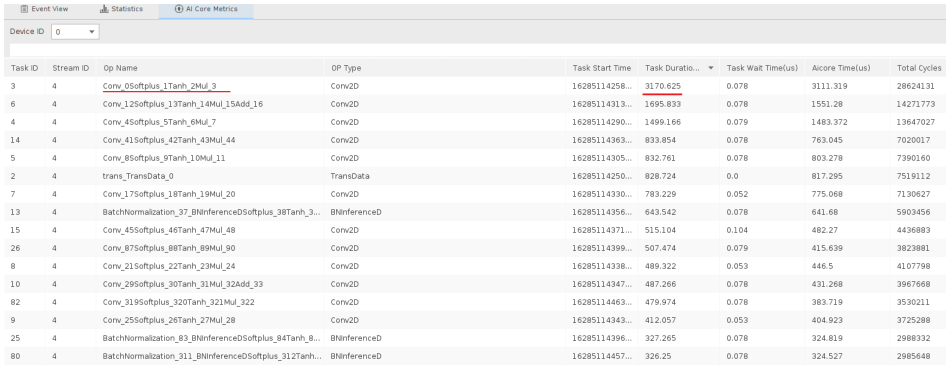
参见《**AscendCL应用软件开发指南（C&C++）**》中的“AscendCL API参考”章节查找aclmdlLoadFromFileWithMem接口的作用为“从文件加载离线模型数据”，可以分析该接口耗时取决于加载离线模型的时间，加载时间我们暂时无法进行调优。

步骤4 继续查找aclmdlExecute接口的作用为“执行应用推理，直到返回推理结果，同步接口”。

可以发现该接口是执行接口，也就是说应用在执行推理的过程中确实存在耗时长的问题，而模型中所有算子执行的时间总和就是执行耗时。

那么查看Profiling结果**10.8.2.5 AI Core Metrics视图**中的算子执行耗时并按Task Duration算子执行任务的时间从高到低排列，如图10-59所示。

图 10-59 AI Core Metrics



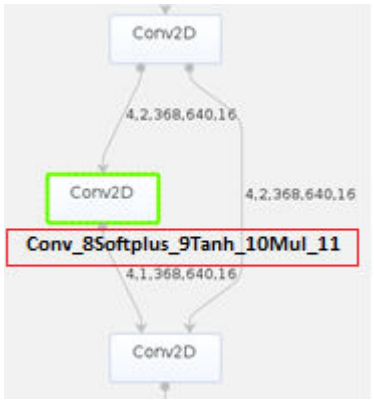
Task ID	Stream ID	Op Name	OP Type	Task Start Time	Task Duratio...	Task Wait Time(us)	Aicore Time(us)	Total Cycles
3	4	Conv_0Softplus_1Tanh_2Mul_3	Conv2D	16285114258...	3170.625	0.078	3111.319	28624131
6	4	Conv_12Softplus_13Tanh_14Mul_15Add_16	Conv2D	16285114313...	1695.833	0.078	1551.28	14271773
4	4	Conv_4Softplus_5Tanh_6Mul_7	Conv2D	16285114290...	1499.166	0.079	1483.372	13647027
14	4	Conv_41Softplus_42Tanh_43Mul_44	Conv2D	16285114363...	833.854	0.078	763.045	7020017
5	4	Conv_8Softplus_9Tanh_10Mul_11	Conv2D	16285114305...	832.761	0.078	803.278	7390160
2	4	trans_TransData_0	TransData	16285114250...	828.724	0.0	817.295	7519112
7	4	Conv_17Softplus_18Tanh_19Mul_20	Conv2D	16285114330...	783.229	0.052	775.068	7130627
13	4	BatchNormalization_37_BNInferenceDSoftplus_38Tanh_3...	BNInferenceD	16285114356...	643.542	0.078	641.68	5903456
15	4	Conv_45Softplus_46Tanh_47Mul_48	Conv2D	16285114371...	515.104	0.104	482.27	4436883
26	4	Conv_87Softplus_88Tanh_89Mul_90	Conv2D	16285114399...	507.474	0.079	415.639	3823881
8	4	Conv_21Softplus_22Tanh_23Mul_24	Conv2D	16285114338...	489.322	0.053	446.5	4107798
10	4	Conv_29Softplus_30Tanh_31Mul_32Add_33	Conv2D	16285114347...	487.266	0.078	431.268	3967668
82	4	Conv_319Softplus_320Tanh_321Mul_322	Conv2D	16285114463...	479.974	0.078	383.719	3530211
9	4	Conv_25Softplus_26Tanh_27Mul_28	Conv2D	16285114343...	412.057	0.053	404.923	3725288
25	4	BatchNormalization_83_BNInferenceDSoftplus_84Tanh_8...	BNInferenceD	16285114396...	327.265	0.078	324.819	2988332
80	4	BatchNormalization_311_BNInferenceDSoftplus_312Tanh...	BNInferenceD	16285114457...	326.25	0.078	324.527	2985648

可以看到第一个Conv算子的执行时间为3170.625us远高于同进程下其他算子的执行时间，可以判断该函数拖慢了整体的执行效率。

到此Profiling性能分析工具的任务已经完成。

步骤5 接下来可以通过**6.5 模型可视化**工具打开PyTorch网络转换成的om模型查询Conv算子，发现该算子是多个计算单元组成，如**图10-60**所示，这样会造成极大的推理开销。

图 10-60 PyTorch 网络 om 模型



步骤6 通过查询代码发现计算单元中的Softplus、Tanh和Mul是属于Mish激活函数的计算公式，如**图10-61**所示。

当前昇腾平台支持的激活函数只有：Relu、Leakyrelu、Prelu、Elu和Srelu，Mish函数不在支持范围内，因此造成模型转换后的Mish函数被分解成了多个计算单元。

图 10-61 Mish 激活函数

```
class Mish(nn.Module):
    def __init__(self):
        super().__init__()
        print("Mish activation loaded...")

    def forward(self, x):
        x = x * (torch.tanh(F.softplus(x)))
        return x
```

解决该问题最简单的办法就是找到效率更高的替代函数。

----结束

问题解决

尝试以昇腾官方提供的Leaky Relu激活函数作为替换函数。函数替换操作请用户自行处理，此处不作阐述。

完成函数替换后重新执行**Profiling性能分析操作**得到新的结果，AI Core Metrics视图下Leaky Relu函数的第一个Conv算子的执行时间Task Duration为1415.182us，比之前Mish函数的第一个Conv算子的执行时间3170.625us降低了一半，同时Leaky Relu函数的精度比Mish函数要小1%，Leaky Relu函数精度更高。如**图10-62**所示。

图 10-62 应用推理 Leaky Relu 函数算子运行结果

Task ID	Stream ID	Op Name	OP Type	Task Start Time	Task Duratio...	Task Wait Time(us)	Aicore Time(us)	Total Cycles
3	4	Conv_0LeakyRelu_1	Conv2D	17020658102...	1415.182	0.078	1384.101	12733727
4	4	Conv_2LeakyRelu_3	Conv2D	17020658117...	1043.828	0.078	1035.973	9530956
6	4	Conv_6LeakyRelu_7Add_8	Conv2D	17020658132...	879.843	0.079	828.237	7619785
2	4	trans_TransData_0	TransData	17020658094...	834.454	0.0	822.426	7566316
7	4	Conv_9LeakyRelu_10	Conv2D	17020658141...	626.693	0.052	625.443	5754077
5	4	Conv_4LeakyRelu_5	Conv2D	17020658127...	515.078	0.052	498.699	4588033
15	4	Conv_25LeakyRelu_26	Conv2D	17020658166...	439.896	0.078	439.291	4041478
27	4	Conv_51LeakyRelu_52	Conv2D	17020658182...	379.037	0.078	377.169	3469955
14	4	Conv_23LeakyRelu_24	Conv2D	17020658162...	356.745	0.078	352.321	3241351

结论

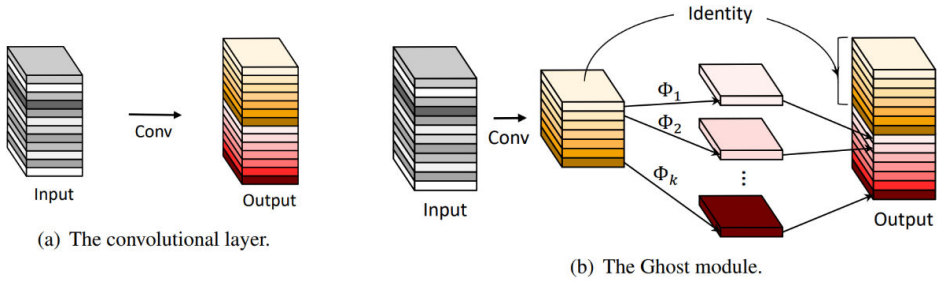
通过Profiling性能分析工具前后两次对网络应用推理的运行时间进行分析，并对比两次执行时间可以得出结论，替换Leaky Relu激活函数后，降低了Conv算子在应用推理的运行时间，提升了推理效率。

10.9.2 网络模型优劣选择分析样例

背景介绍

以Ghostnet网络模型为例，Ghostnet网络模型宣称为轻量化网络模型，在网络应用推理效率上高于传统卷积的网络模型。因此使用Profiling性能分析工具对网络应用执行推理耗时分析，分析结果显示推理耗时数值较高。通过对该模型的卷积操作进行分析，发现Ghostnet网络模型的Conv操作进行了多次拆分整合，该操作严重影响了运行效率，并不如普通网络模型的推理效率高。如**图10-63**所示。

图 10-63 Ghostnet 网络与传统卷积的网络模型的卷积操作对比



得出结论：Ghostnet网络模型不是优选的网络模型。

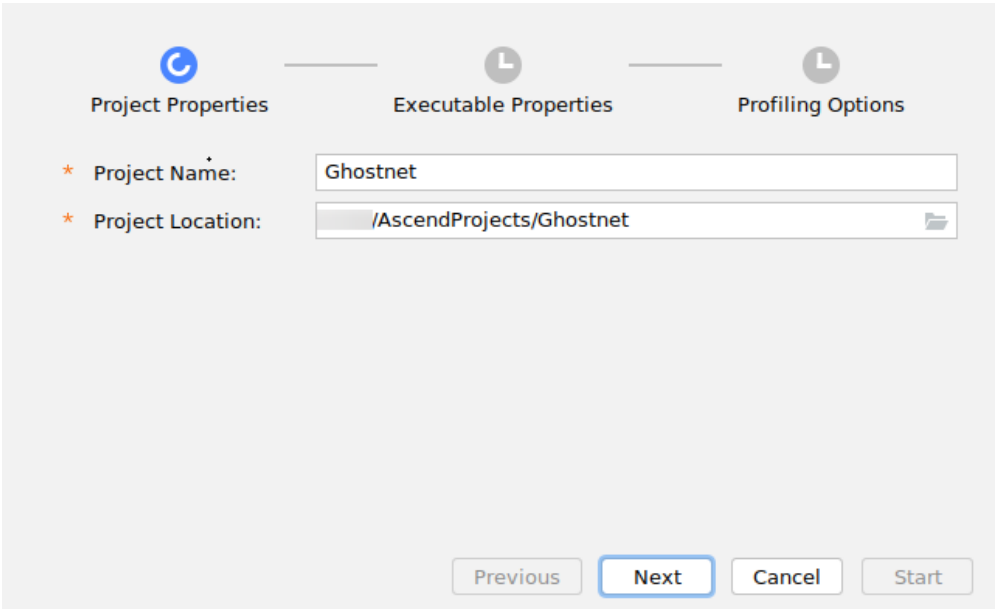
Profiling 性能分析操作

步骤1 启动MindStudio，单击选择并打开已编译完成的工程。

步骤2 单击菜单栏“Ascend > System Profiler > New Project”，弹出Profiling配置窗口 。

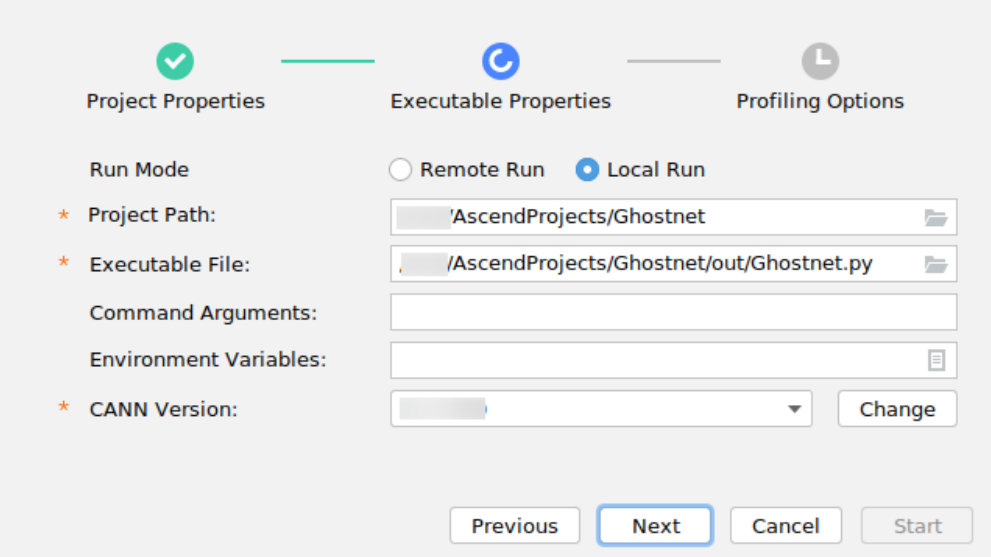
步骤3 进入Profiling配置窗口，如图10-64。配置Profiling的工程名称“Project Name”和选择Profiling工程的结果路径“Project Location”。单击“Next”进入下一步。

图 10-64 Project Properties 配置



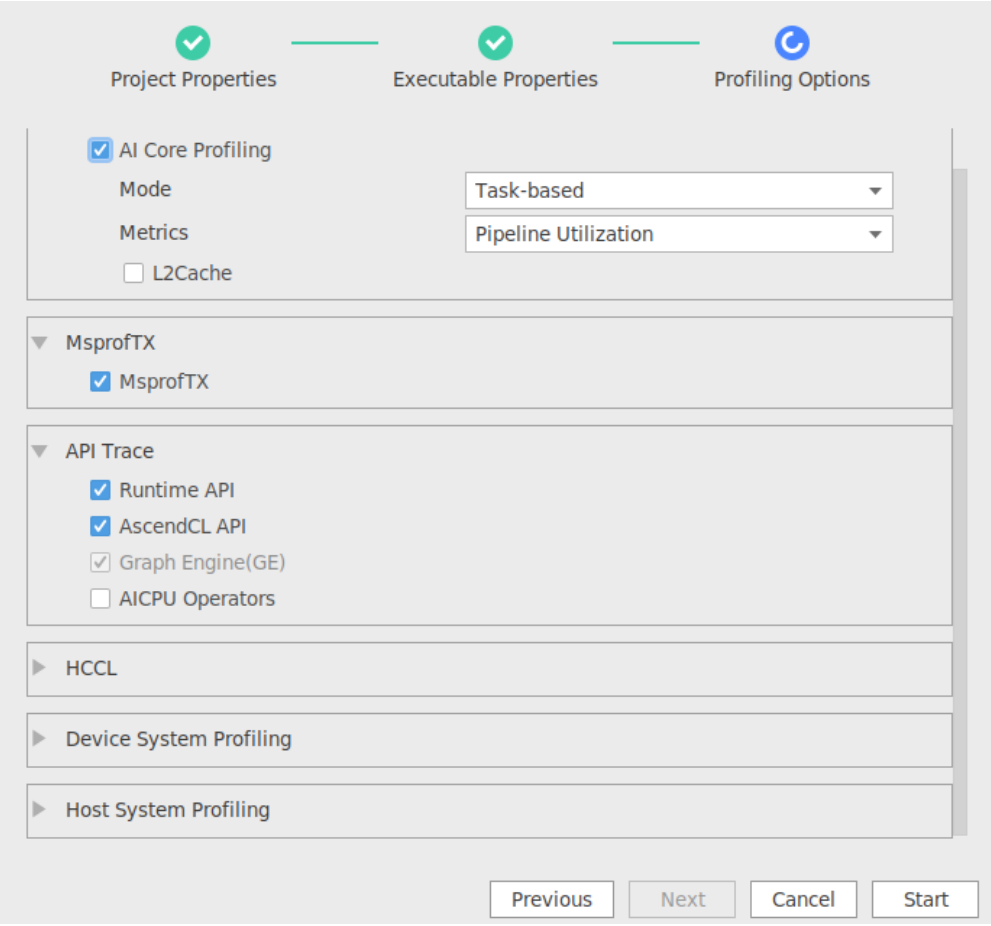
步骤4 进入“Executable Properties”配置界面，选择Local Run模式。指定执行Profiling目标工程的可执行文件目录。如图10-65所示。

图 10-65 Local Run



步骤5 进入“Profiling Options”配置界面，选择Task-based场景。如图10-66所示。

图 10-66 Task-based 场景



步骤6 完成上述配置后单击窗口右下角的“Start”按钮，启动Profiling。

工程执行完成后，MindStudio窗口下方自动展示Profiling结果视图。单击Timeline视图下方数据窗格的Statistics视图中aclmdlExecute接口体现网络整体运行时间达到了66352.469us，单击AI Core Metrics视图，查看AI Core Metrics数据中Ghostnet网络模型中执行了大量的Concat操作，并且运行耗时均较长。如**图10-67**和**图10-68**所示。

图 10-67 Ghostnet 网络应用推理运行结果_Statistics

Process ID	Thread ID	Type	Name	Time(%)	Time(us)	Count	Avg(us)	Max(us)	Min(us)
16784	16784	model	aclmdlLoadFromFileWithMem	51.855909	174343.974	1	174343.974	174343.974	174343
16784	16784	model	aclmdlExecute	19.735512	66352.469	1	66352.469	66352.469	66352.4
16784	16784	model	aclmdlQuerySize	9.07647	30515.863	1	30515.863	30515.863	30515.8
16784	16784	runtime	aclrtFree	8.419892	28308.391	4	7077.09775	17273.393	345.022
16784	16784	runtime	aclrtFreeHost	2.781378	9351.23	2	4675.615	6813.572	2537.65
16784	16784	runtime	aclrtMallocHost	2.232927	7507.292	2	3753.646	6787.274	720.018
16784	16784	model	aclmdlUnload	2.013371	6769.126	1	6769.126	6769.126	6769.12
16784	16784	runtime	aclrtMalloc	1.879446	6318.857	4	1579.71425	2338.71	546.458
16784	16784	runtime	aclrtMemcpy	1.485234	4993.482	2	2496.741	3400.054	1593.42
16784	16784	runtime	aclrtCreateContext	0.271518	912.868	1	912.868	912.868	912.868
16784	16784	runtime	aclrtDestroyContext	0.139009	467.359	1	467.359	467.359	467.359
16784	16784	runtime	aclrtCreateStream	0.080359	270.173	1	270.173	270.173	270.173
16784	16784	model	aclmdlGetDesc	0.018338	61.654	1	61.654	61.654	61.654
16784	16784	runtime	aclrtDestroyStream	0.010636	35.76	1	35.76	35.76	35.76

图 10-68 Ghostnet 网络应用推理运行结果_AI Core Metrics

Task ID	Stream ID	Op Name	OP Type	Task Start Time	Task Duratio...	Task Wait Time(us)	Aicore Time(us)	Total Cycles
6	2	Conv_5LeakyRelu_6	Conv2D	43520979473...	1802.891	0.078	1765.975	16246968
3	2	Conv_0LeakyRelu_1	Conv2D	43520973494...	1409.896	0.078	1388.104	12770554
14	2	Concat_17	ConcatD	43520987195...	1369.584	0.078	1366.099	12568114
8	2	Concat_9	ConcatD	43520982175...	1368.255	0.078	1365.587	12563396
16	2	Conv_19LeakyRelu_20	Conv2D	43520990639...	1210.729	0.078	1205.427	11089930
32	2	Add_41	Add	43520999231...	1040.651	0.078	1035.914	9530410
25	2	Add_32	Add	43520995609...	1040.313	0.078	1036.689	9537541
13	2	Conv_16	Conv2D	43520986281...	914.479	0.104	896.031	8243485
7	2	Conv_7LeakyRelu_8	Conv2D	43520981276...	898.724	0.078	884.708	8139312
9	2	Conv_10LeakyRelu_11	Conv2D	43520983543...	891.511	0.078	883.959	8132419
33	2	Conv_42LeakyRelu_43	Conv2D	43521000272...	842.265	0.079	822.657	7568440
2	2	trans_TransData_0	TransData	43520972665...	828.75	0.0	815.804	7505398
92	2	Conv_119LeakyRelu_120	Conv2D	43521016043...	790.573	0.078	789.141	7260099
248	2	Conv_324LeakyRelu_325	Conv2D	43521036713...	746.041	0.105	733.517	6748358
24	2	Concat_31	ConcatD	43520994924...	685.365	0.078	681.477	6269592
18	2	Concat_23	ConcatD	43520992354...	685.052	0.105	682.106	6275371
31	2	Concat_40	ConcatD	43520998547...	684.114	0.078	681.261	6267604

----结束

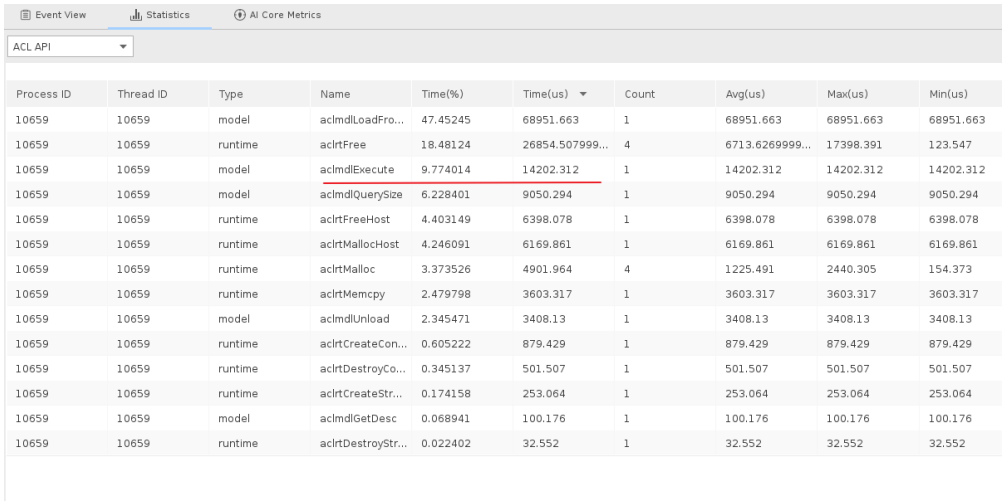
问题分析

根据**图10-68**所示的结果，Ghostnet网络模型因为执行了大量的Concat操作导致了整体的运行时间较高。

问题解决

使用普通网络模型重新执行**Profiling性能分析操作**得到新的结果Statistics视图中aclmdlExecute接口体现网络整体运行时间只有14202.312us。如**图10-69**所示。

图 10-69 普通网络模型运行结果


Event View Statistics AI Core Metrics

ACL API

Process ID	Thread ID	Type	Name	Time(%)	Time(us)	Count	Avg(us)	Max(us)	Min(us)
10659	10659	model	aclmdlLoadFro...	47.45245	68951.663	1	68951.663	68951.663	68951.663
10659	10659	runtime	aclrtFree	18.48124	26854.507999...	4	6713.6269999...	17398.391	123.547
10659	10659	model	aclmdlExecute	9.774014	14202.312	1	14202.312	14202.312	14202.312
10659	10659	model	aclmdlQuerySize	6.228401	9050.294	1	9050.294	9050.294	9050.294
10659	10659	runtime	aclrtFreeHost	4.403149	6398.078	1	6398.078	6398.078	6398.078
10659	10659	runtime	aclrtMallocHost	4.246091	6169.861	1	6169.861	6169.861	6169.861
10659	10659	runtime	aclrtMalloc	3.373526	4901.964	4	1225.491	2440.305	154.373
10659	10659	runtime	aclrtMemcpy	2.479798	3603.317	1	3603.317	3603.317	3603.317
10659	10659	model	aclmdlUnload	2.345471	3408.13	1	3408.13	3408.13	3408.13
10659	10659	runtime	aclrtCreateCon...	0.605222	879.429	1	879.429	879.429	879.429
10659	10659	runtime	aclrtDestroyCo...	0.345137	501.507	1	501.507	501.507	501.507
10659	10659	runtime	aclrtCreateStr...	0.174158	253.064	1	253.064	253.064	253.064
10659	10659	model	aclmdlGetDesc	0.068941	100.176	1	100.176	100.176	100.176
10659	10659	runtime	aclrtDestroyStr...	0.022402	32.552	1	32.552	32.552	32.552

结论

通过Profiling性能分析工具前后两次对网络应用推理的运行时间进行分析，并对比两次执行时间可以得出结论，Ghostnet网络应推理效率并不比传统卷积的网络应用推理效率高，可以不作为优先选择的网络模型。

10.9.3 自定义算子性能优化分析样例

背景介绍

算子是组成模型的基本单元，而自定义算子往往存在优化空间。

为了验证自定义算子Vadd_sample的性能，基于向量加法构建了复杂样例算子Vadd_sample，并使用Profiling工具对其性能进行分析。由于Vadd_sample是在向量加法基础上，构建的一个复杂算子，只在Vector上执行，因此算子的性能指标可通过AI Core Metrics视图下的vec_ratio（向量类运算指令的cycle数在所有指令的cycle数中的占用比）字段来判断。查看分析结果中Vadd_sample算子的vec_ratio指标，发现数值较低，未达到最优性能。由此可以得出结论，自定义算子Vadd_sample未达到最优性能并存在优化空间。

说明

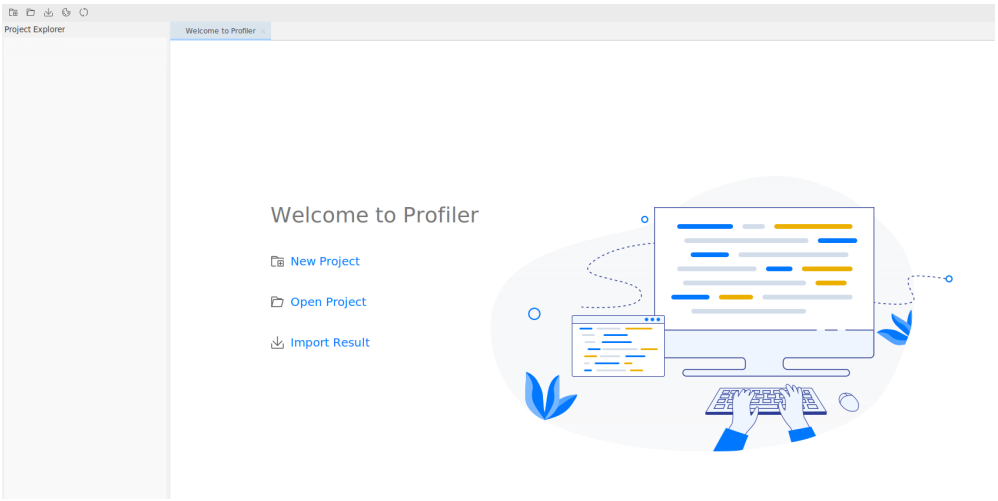
- 本文仅介绍Profiling性能分析工具的操作及分析过程，对于自定义算子的详细优化编程此处不做阐述。
- 有关自定义算子的详细介绍请参见**8 算子开发**章节。

Profiling 性能分析操作

步骤1 启动MindStudio，单击选择并打开已编译完成的工程。

步骤2 单击菜单栏“Ascend > System Profiler”，弹出系统分析工程界面。

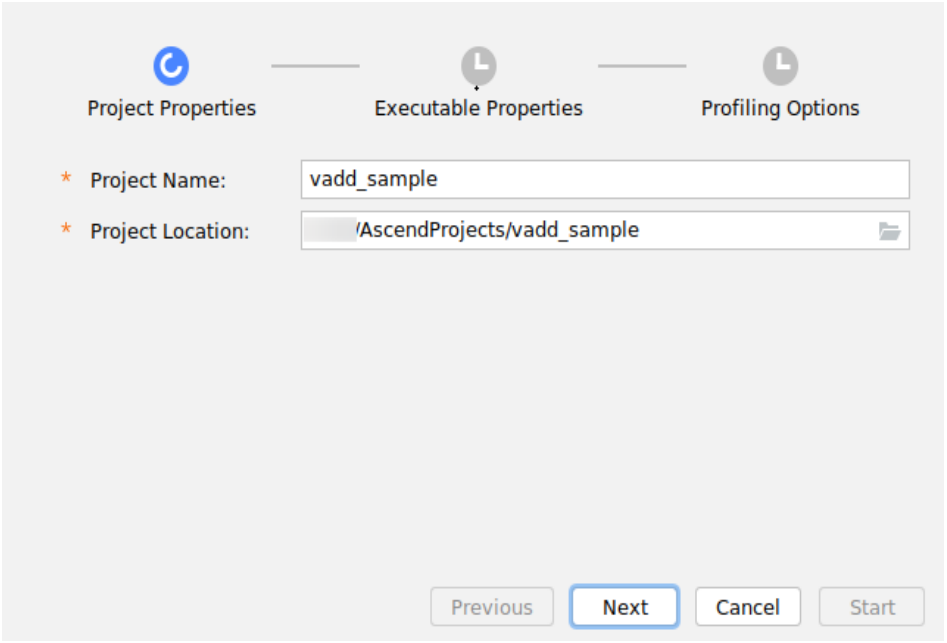
图 10-70 系统分析工程界面



步骤3 进入系统分析工程界面，单击欢迎界面中的“New Project”或左上角的 图标，弹出Profiling配置窗口，如图10-71。

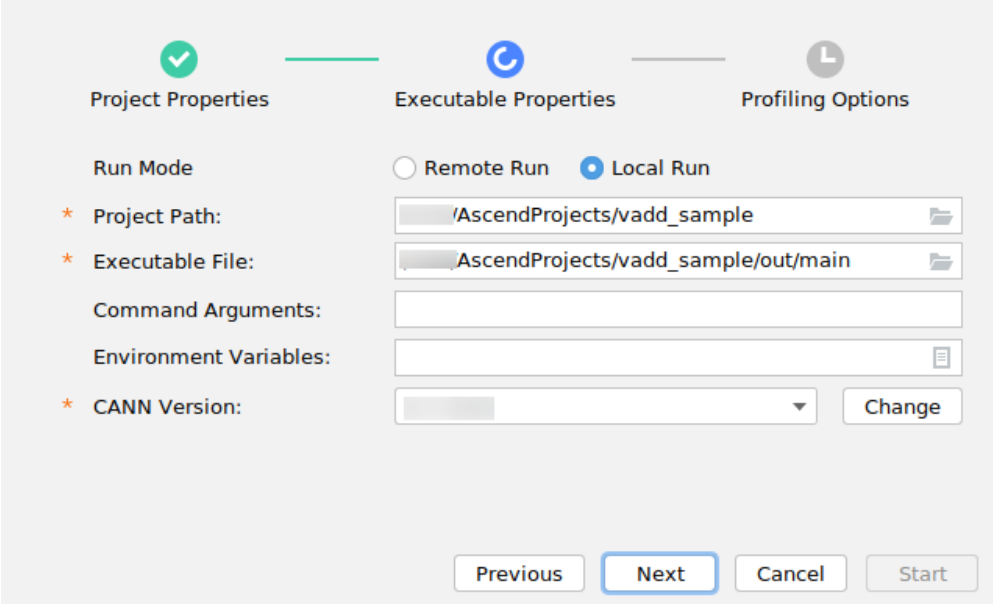
配置Profiling的工程名称“Project Name”和选择Profiling工程的结果路径“Project Location”。单击“Next”进入下一步。

图 10-71 Project Properties 配置



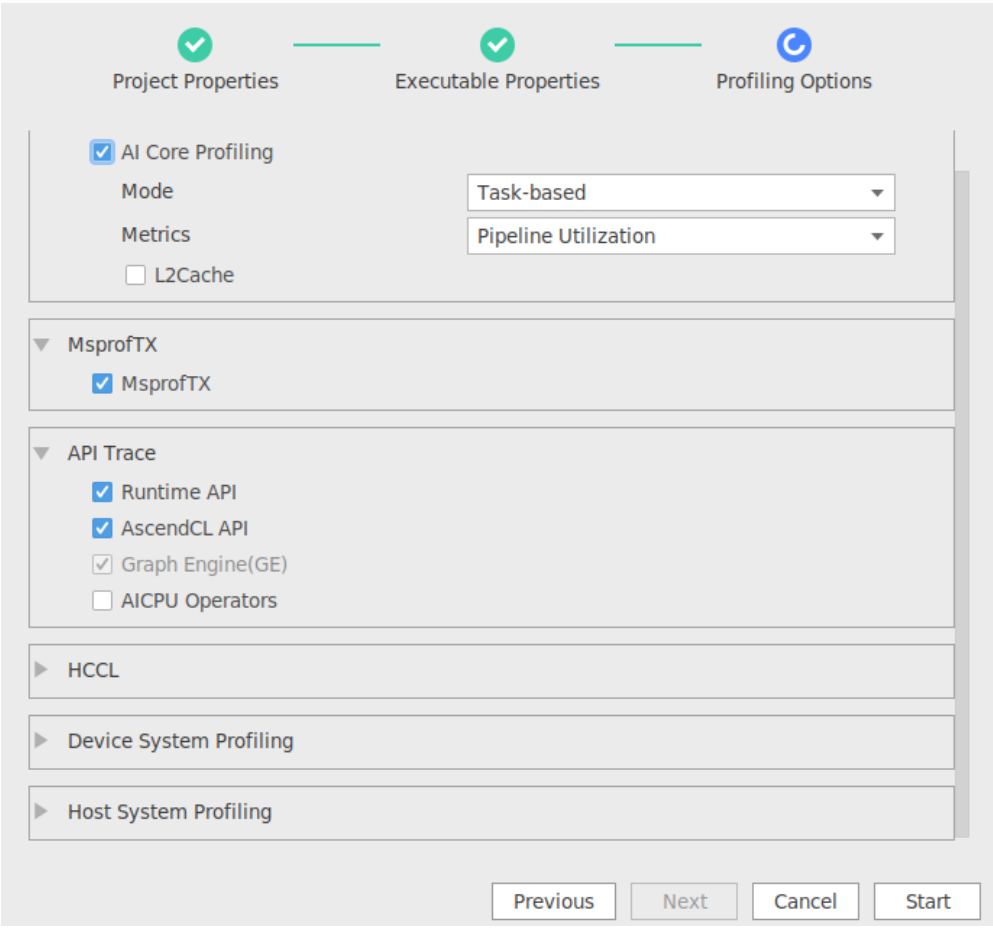
步骤4 进入“Executable Properties”配置界面，选择Local Run模式。指定执行Profiling目标工程的可执行文件目录。如图10-72所示。

图 10-72 Local Run



步骤5 进入“Profiling Options”配置界面，选择Task-based场景。如图10-73所示。

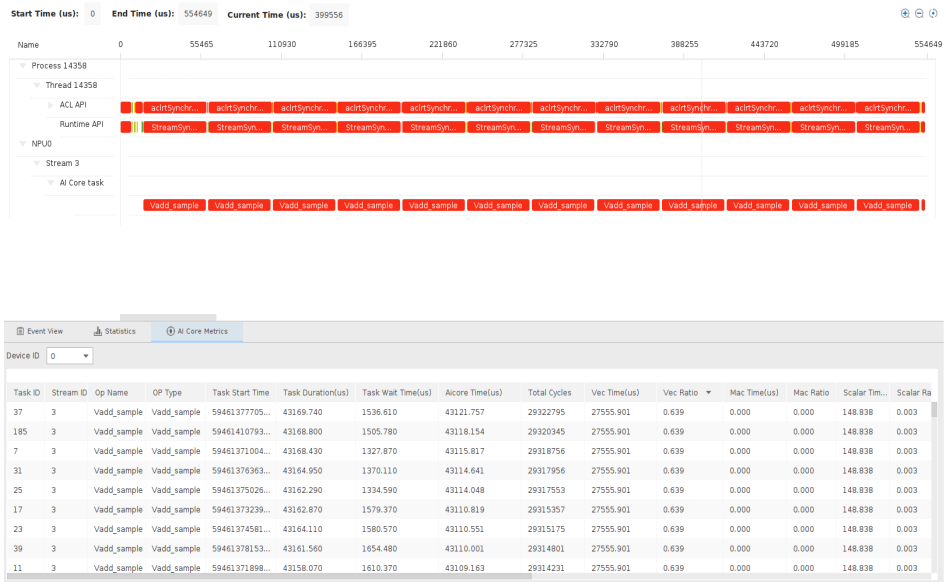
图 10-73 Task-based 场景



步骤6 完成上述配置后单击窗口右下角的“Start”按钮，启动Profiling。

工程执行完成后，MindStudio窗口下方自动展示Profiling结果视图。如图10-74所示。

图 10-74 Profiling 分析结果



----结束

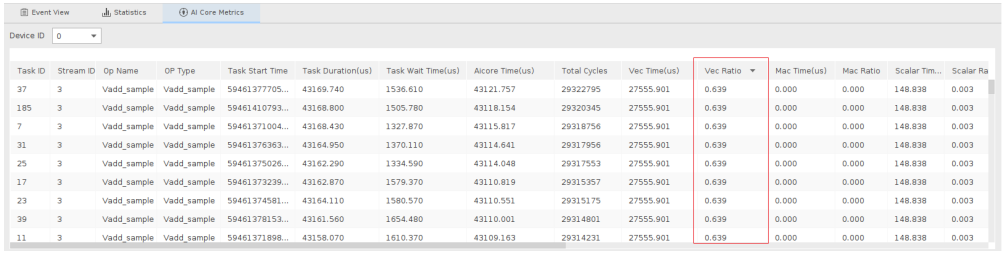
问题分析

首先，用户可通过查看Timeline，感知应用的整体运行情况。然后结合调色盘功能，可方便地标记和发现耗时超标的API/算子，结合API/算子的统计表格（Statistics）可初步定位潜在瓶颈。最后，通过观测算子的AI Core Metrics即分析算子是否存在性能不足和优化空间，可以最终确定优化切入点。

在本例中，由于只有一个算子对大量数据做处理，因而可直接切入分析AI Core Metrics。Vadd_sample是一个计算密集的Vector算子，因此需要关注vec_ratio（向量类运算指令的cycle数在所有指令的cycle数中的占用比，代表着资源利用率）是否达标。如图10-75所示。

不同算子类型需要关注不同指标，如重搬运轻计算的算子，则更应关注访存性能，根据实际情况分析即可。

图 10-75 AI Core Metrics



可以看到AI Core Metrics视图中，Vadd_sample的vec_ratio最大仅达到0.639，而vec_ratio指标取值越接近1资源利用率越高，故此可以判断Vadd_sample算子存在可优化空间。

到此Profiling性能分析工具的任务完成。

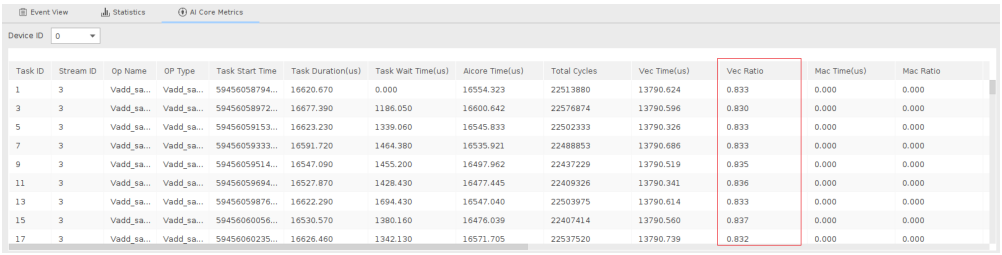
问题解决

为了优化Vadd_sample算子，考虑结合“TIK优化指南”对Vadd_sample算子采取双核并行、double buffer等优化方式，提升Vadd_sample的Vector计算资源利用率，最终减少算子执行耗时。

具体编程操作请开发者自行完成。

完成Vadd_sample算子的优化后，对推理应用重新执行**Profiling性能分析操作**得到新的结果，再次查看AI Core Metrics视图，如**图10-76**所示。

图 10-76 AI Core Metrics



Task ID	Stream ID	Op Name	OP Type	Task Start Time	Task Duration(us)	Task Wait Time(us)	Aicore Time(us)	Total Cycles	Vec Time(us)	Vec Ratio	Mac Time(us)	Mac Ratio
1	3	Vadd_sa...	Vadd_sa...	59456058794...	16620.670	0.000	16554.323	22513880	13790.624	0.833	0.000	0.000
3	3	Vadd_sa...	Vadd_sa...	59456058972...	16677.390	1186.050	16600.642	22576874	13790.596	0.830	0.000	0.000
5	3	Vadd_sa...	Vadd_sa...	59456059153...	16623.230	1339.060	16545.833	22502333	13790.326	0.833	0.000	0.000
7	3	Vadd_sa...	Vadd_sa...	59456059333...	16591.720	1464.380	16535.921	22488853	13790.686	0.833	0.000	0.000
9	3	Vadd_sa...	Vadd_sa...	59456059514...	16547.090	1455.200	16497.962	22437229	13790.519	0.835	0.000	0.000
11	3	Vadd_sa...	Vadd_sa...	59456059694...	16527.870	1428.430	16477.445	22409326	13790.341	0.836	0.000	0.000
13	3	Vadd_sa...	Vadd_sa...	59456059876...	16622.290	1694.430	16547.040	22503975	13790.614	0.833	0.000	0.000
15	3	Vadd_sa...	Vadd_sa...	59456060056...	16530.570	1380.160	16476.039	22407414	13790.560	0.837	0.000	0.000
17	3	Vadd_sa...	Vadd_sa...	59456060235...	16626.460	1342.130	16571.705	22537520	13790.739	0.832	0.000	0.000

可以看到优化后AI Core Metrics视图中的vec_ratio指标平均达到0.83左右，说明Vadd_sample的Vector计算资源利用率得到了提升。

结论

通过Profiling性能分析工具前后两次对网络应用推理的资源利用率进行分析，并对比前后算子优化结果可以得出结论，自定义算子Vadd_sample未达到最优性能，存在优化空间。

11 专家系统

11.1 专家系统简介

专家系统（MindStudio Advisor）是用于聚焦模型和算子的性能调优Top问题，识别性能瓶颈，重点构建瓶颈分析、优化推荐模型，支撑开发效率提升的工具。

说明

以下功能需根据**11.3 输入数据说明**完成输入数据准备，其中基于Roofline模型的算子瓶颈识别与优化建议、基于Timeline的AI CPU算子优化、算子融合推荐、TransData算子识别通过**11.4.1 专家系统入口**操作，算子优化分析通过**11.4.2 算子工程入口**操作。

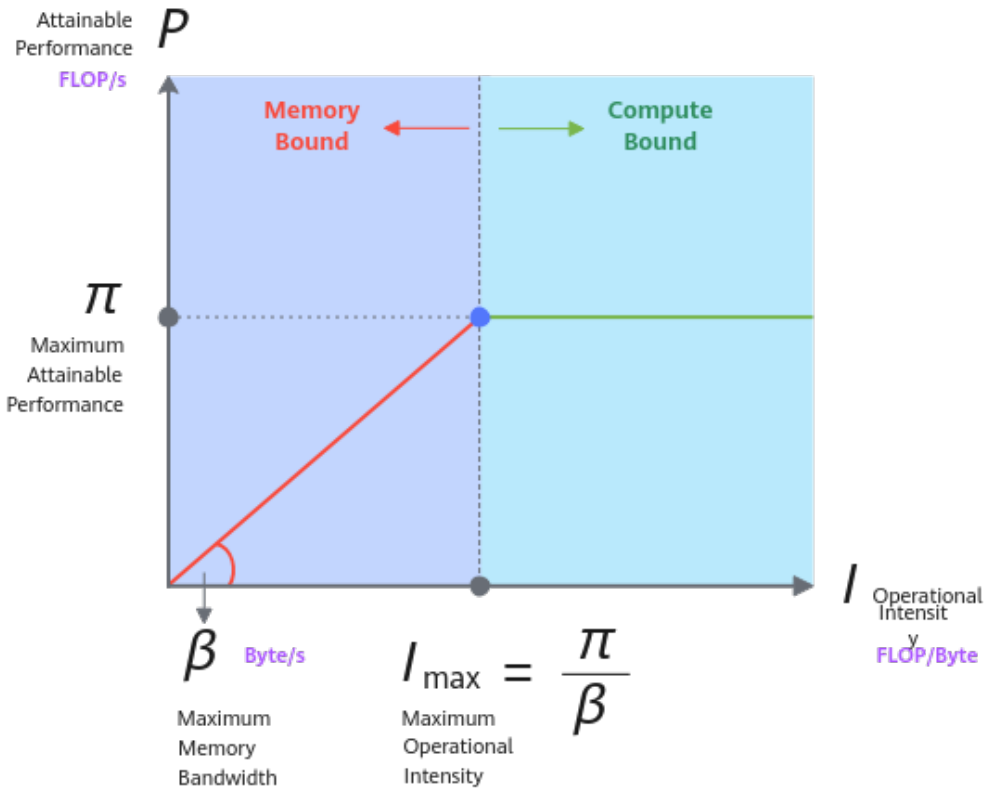
基于 Roofline 模型的算子瓶颈识别与优化建议

如**图11-1**所示，横坐标单位是FLOP/Byte，表示计算强度，每搬运1Byte数据可以进行多少次运算，越大表示内存搬运利用率越高。纵坐标单位是FLOP/s，表示运算速度，越大表示运算越快。

当横坐标很大时，表示每搬运1byte数据可以进行很多的运算，但是每秒的运算次数无法超过硬件的性能上限π，即图中绿色横线；随着横坐标减小到一定的阈值以下，即图中蓝点的横坐标Imax，搬运的数据将不支持硬件达到性能上限的算力，此时的纵坐标性能为β·x，β表示硬件的带宽，即图中红色斜线。

蓝点将Roofline模型分成2个部分，红色部分为Memory Bound，绿色部分为Compute Bound，且实际工作点越远离红线或绿线，表示Bound越严重，为主要瓶颈所在。

图 11-1 roofline 模型介绍



基于 Timeline 的 AI CPU 算子优化

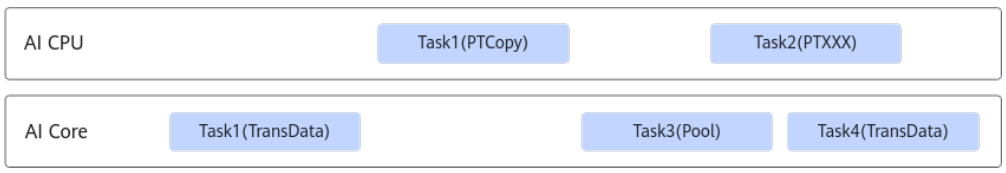
AI CPU是昇腾AI处理器计算单元，因为该CPU计算处理单位自身瓶颈，导致运行在AI CPU上的算子影响模型的执行时间。AI CPU算子的优化往往是重点关注点和优化对象。

模型开发和模型转换过程中，引入AI CPU算子，出现因为串行等待AI CPU算子执行影响模型执行。

从时间序列分析，性能瓶颈一般由串行等待算子造成。当前Timeline时序信息以Stream粒度展示，无法直观发现算子间的串并行关系。

如图11-2，AI CPU Timeline中Task1（PTCopy）存在模型执行串行等待AI CPU算子执行，瓶颈分析模型需要主动识别这类瓶颈。AI CPU Timeline中Task2计算时间隐藏在AI Core计算时间中，这类AI CPU算子执行可以忽略。

图 11-2 AI CPU 算子执行



基于Timeline的AI CPU算子优化以Profiling Task Scheduler任务调度信息数据（task_time_xxxx.json）以及OM离线模型文件作为输入数据，自动识别串行执行AI CPU算子，给出优化建议，提升模型整体性能。

算子融合推荐

算子融合推荐包括UB算子融合、首层算子融合和L2融合（动态Batch切分）三个功能。

UB算子融合

模型转换过程中，由于多种原因，如算子计算的数据超过UB大小 ，当前算子融合规则未覆盖等，会造成算子融合失败。当OM模型非常复杂时，如包含几千个计算节点时，靠肉眼定位到可融合算子耗时耗力。模型转换工具规则不能匹配所有场景，融合规则有遗漏。

UB算子融合推荐以OM离线模型文件作为输入数据，自动发现OM模型中的可融合算子，发现算子融合遗漏的场景，提供算子融合建议。

首层算子融合

在load3dv2支持输入通道非16/32对齐的场景，可以直接支持4通道卷积，极大减少MTE2与cube运算。该模式在非AIPP场景下，算子首层需要cast和trans_data算子进行图片的数据类型转换和数据排布格式转换，在C0=4的场景下，trans_data算子性能恶化严重，此时整网性能恶化严重。若改为AIPP场景，此时网络首层会进行AIPP+conv融合或AIPP+conv+maxpooling融合，由于一般首层16通道卷积的耗时占比整网的10%左右，此时开启small channel模式会大幅提升整网性能。

首层算子融合以OM离线模型文件作为输入数据，自动发现OM模型中可优化的数据预处理算子，使能AIPP，提升性能。

L2融合（动态Batch切分）

在整网中，由于某些算子层为L2融合，数据量较大（包含输入和输出数据），超出L2的空间，会引发DDR（DDR主要统计读写带宽，在Analysis Summary中以表格形式呈现）写回的操作；而DDR的带宽远小于L2，造成MTE2 bound严重，同时可能进一步引发流水问题。

以resnet50_int8_8batch为例：

理论上8batch的单算子计算耗时小于等于2倍4batch单算子计算耗时，但如果8batch各层的算子为L2融合，导致数据量变大，造成L2 cache空间不足，产生DDR写回情况，引发算子性能恶化。

表 11-1 8batch_4batch 算子计算性能对比

算子	8batch性能/us	4batch性能/us	8batch/4batch倍数
res2a_branch2c	200.63	38.284	5.24
res2b_branch2c	189.97	40.527	4.69
res2c_branch2c	147.84	38.533	3.84
res3a_branch1	74.00	31.109	2.38
整网时间	2031	954	-
整网占比	30.2%	15.5%	-

如表11-1所示，可以看到8batch的算子计算耗时远超过4batch算子的2倍，算子性能恶化较严重。

专家系统工具通过分析OM模型中的各层算子，以及读取op_summary.csv数据，识别并输出L2融合且未达到性能瓶颈的算子。需要对这些算子进行切分以实现每层算子的数据均不超过L2的范围，防止DDR写回的情况发生。

TransData 算子识别

模型中格式转换是影响模型性能的一个重要因素。当前格式转换由NPU CUBE单元的特性引入，在CV网络主要体现大量的4D与5D的转换，而在NLP网络主要体现为大量4D与NZ的转换。大量的TransData算子会影响模型的性能。

TransData算子识别功能通过分析模型引入TransData的常见场景，识别转化算子性能瓶颈，从算子层、适配层和模型层三个层面进行分析，选择合适的方案进行优化，减少模型中TransData的调用次数。

算子优化分析

算子调优的主要场景：算子开发结束和整网运行过程中出现算子性能不达标。算子调优对开发者的要求比较高，需要开发者对底层和框架有一定的了解，同时具备一定的算子调优经验。算子优化分析可以协助开发者迅速找到算子性能瓶颈，并给出相应的优化手段，能够有效提升开发者算子调优的效率。

通过输入算子仿真生成的dump文件，从向量运算、标量运算、流水打断和Memory bound四个维度进行分析，并给出分析的数据和相应的优化建议。

11.2 芯片支持情况

- 基于Roofline模型的算子瓶颈识别与优化建议：Atlas 200/300/500 推理产品、Atlas 推理系列产品、Atlas 200/500 A2推理产品。
- 基于Timeline的AI CPU算子优化：Atlas 200/300/500 推理产品、Atlas 推理系列产品、Atlas 200/500 A2推理产品。
- 算子融合推荐：Atlas 200/300/500 推理产品、Atlas 推理系列产品、Atlas 200/500 A2推理产品。
- TransData算子识别：Atlas 200/300/500 推理产品、Atlas 推理系列产品、Atlas 200/500 A2推理产品。
- 算子优化分析：Atlas 200/300/500 推理产品。
- Atlas 训练系列产品和Atlas A2 训练系列产品暂不支持专家系统功能。

11.3 输入数据说明

11.3.1 输入数据获取

专家系统当前功能要求准备的输入数据如表11-2所示，用户可以提供其中一种或多种数据的组合，每种瓶颈识别模型需要的数据请参见11.3.2 分析功能与输入数据的对应关系。

表 11-2 专家系统输入数据

输入数据	存储目录	说明
*.om OM离线模型文件	**${data_path}/data/project**	• 可在**11.4.1 专家系统入口**中指定。 • **基于Timeline的AI CPU算子优化**、**算子融合推荐**、**基于Roofline模型的算子瓶颈识别与优化建议**和**TransData算子识别**功能需准备。 • 请确保OM文件大小在1GB以内，否则无法执行专家系统分析。
*.cce cce代码	**${data_path}/data/project**	• 可在**11.4.1 专家系统入口**中指定。 • **基于Roofline模型的算子瓶颈识别与优化建议**功能需准备。 • 请确保单个CCE文件大小在100MB以内，CCE文件总量要求在1GB以内，否则无法执行专家系统分析。
task_time_*.json Profiling Task Scheduler任务调度信息数据	**${data_path}/data/profiling/PROF_*/device_0**	• 启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。 • 作用于**基于Timeline的AI CPU算子优化**和**基于Roofline模型的算子瓶颈识别与优化建议**功能。功能配置请参见**11.4.1 专家系统入口**。 • 请确保Profiling Task Scheduler任务调度信息数据文件大小在100MB以内，否则无法执行专家系统分析。
*.csv Profiling Summary文件	**${data_path}/data/profiling/PROF_*/device_0**	• 启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。 • **算子融合推荐**的**L2融合**和**TransData算子识别**功能需准备。功能配置请参见**11.4.1 专家系统入口**。 • 请确保Profiling Summary单个文件大小在100MB以内，其中op_Summary文件的总量要求在1GB以内，否则无法执行专家系统分析。

输入数据	存储目录	说明
info.json.0 Profiling基本信息文件	**${data_path}/data/profiling/PROF_*/device_0**	• 启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。 • **TransData算子识别**功能需准备。功能配置请参见**11.4.1 专家系统入口**。 • 请确保Profiling基本信息文件大小在100MB以内，否则无法执行专家系统分析。
core*{id}*_instr_popped_log.dump core*{id}*_instr_log.dump 算子仿真文件，支持tmmodel，其中*{id}*是core的编号	**{project_location}/out/model/tm/{opname}/{casename}**	• 自动识别算子工程中的路径，无需手动指定。 • 作用于**算子优化分析**功能。功能配置请参见**11.4.2 算子工程入口**。 • 请确保算子仿真文件大小在100MB以内，否则无法执行专家系统分析。
Executable File 执行专家系统目标工程的可执行文件	• **C++：{project_location}/out** • **Python：{project_location}/src**	• 可在**11.4.1 专家系统入口**中指定。 • 需要对用户应用下的模型进行精准分析，指定**Inference App模式**时需要准备此文件。 • 获取方式请参见**7.3.6.1 Linux场景编译运行**。
注：${data_path}为数据目录根路径。		

11.3.2 分析功能与输入数据的对应关系

专家系统工具一次执行输出所有子功能的分析结果，但不同功能要求的输入数据不同，本章节介绍分析功能与输入数据的对应关系。如**表11-3**所示。

表 11-3 分析功能与输入数据的对应关系

瓶颈识别功能	文件	获取方式
基于Roofline模型的算子瓶颈识别与优化建议	cce代码	可通过ATC工具添加--op_debug_level=4得到，或用户自行在算子编译时保存。参见《**ATC工具使用指南**》。
	Profiling Task Scheduler任务调度信息数据文件	启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。

瓶颈识别功能	文件	获取方式
	OM离线模型文件	可通过多种方式生成，例如ATC工具转换，参见**6 模型转换和调优**。
基于Timeline的AI CPU算子优化	Profiling Task Scheduler任务调度信息数据文件	启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。
	OM离线模型文件	可通过多种方式生成，例如ATC工具转换，参见**6 模型转换和调优**。
算子融合推荐	Profiling Summary文件	使用op_Summary.csv文件和l2_cache.csv文件，启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。
	OM离线模型文件	可通过多种方式生成，例如ATC工具转换，参见**6 模型转换和调优**。
TransData算子识别	Profiling Summary文件、Profiling基本信息文件（主要获取文件中当前芯片版本的基本信息）	参见启动专家系统功能后自动拉起Profiling获取数据，无需手动配置。
	OM离线模型文件	可通过多种方式生成，例如ATC工具转换，参见**6 模型转换和调优**。
算子优化分析	算子仿真文件	通过tmmodel仿真得到。参见**8 算子开发**完成算子工程的创建和开发，运行算子UT测试即可，无需手动获取文件。

输入数据文件名及保存路径请参见**11.3.1 输入数据获取**。

11.4 操作步骤

11.4.1 专家系统入口

11.4.1.1 概述

专家系统功能的配置入口包括专家系统入口和算子工程入口。

本节主要介绍专家系统入口的配置方式，该入口可执行的专家系统分析功能包括：**基于Roofline模型的算子瓶颈识别与优化建议**、**基于Timeline的AI CPU算子优化**、**算子融合推荐**和**TransData算子识别**功能。

说明

专家系统自有知识库仅提供模型或算子的可优化项并给出优化建议，具体优化方式请开发者自行修改代码。

11.4.1.2 配置步骤

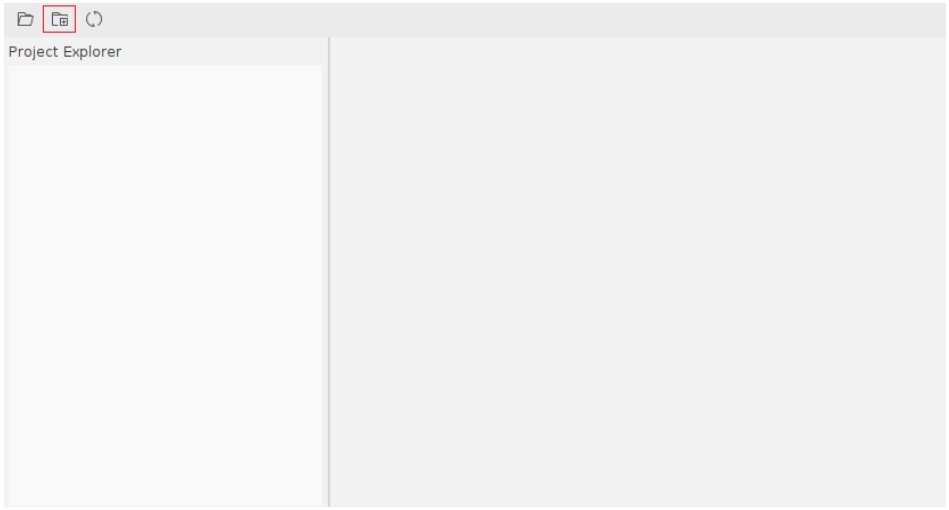
通过以下步骤进入专家系统工具进行配置和分析：

步骤1 选择并打开已编译完成的应用工程。

支持C/C++和Python应用工程，其中Python应用工程不需要编译。

步骤2 单击菜单栏“Ascend > Advisor”，弹出专家系统工具界面 。如图11-3所示。

图 11-3 专家系统工具界面



步骤3 单击图11-3界面左上角“New Project” 按钮，打开专家系统配置界面，可选择Remote Run和Local Run模式且每种模式下可选择OM only和Inference App专家系统分析模式。如下各图所示。

图 11-4 Remote Run OM only（Linux）

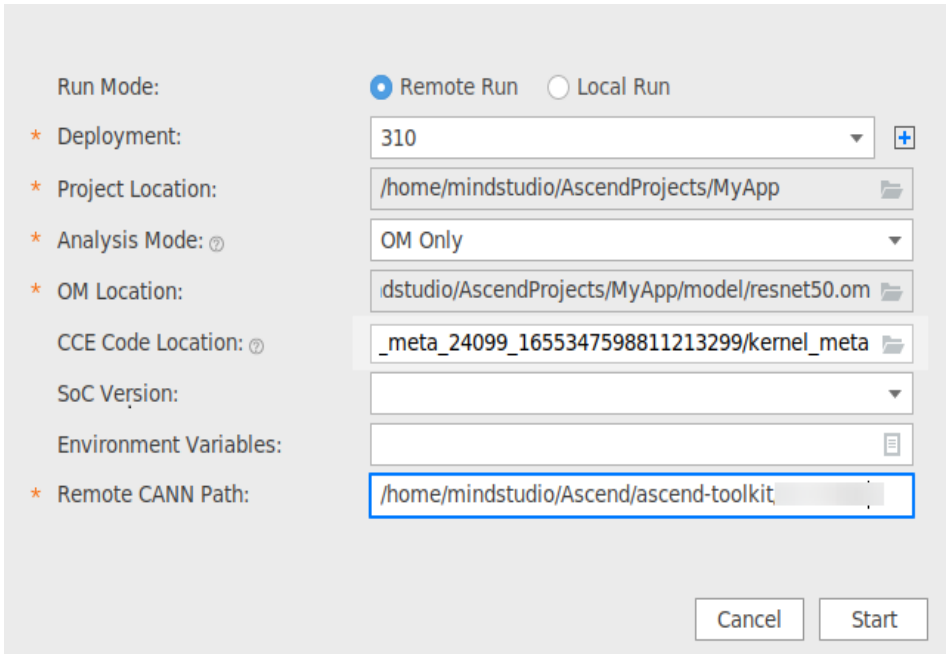


图 11-5 Remote Run Inference App（Linux）

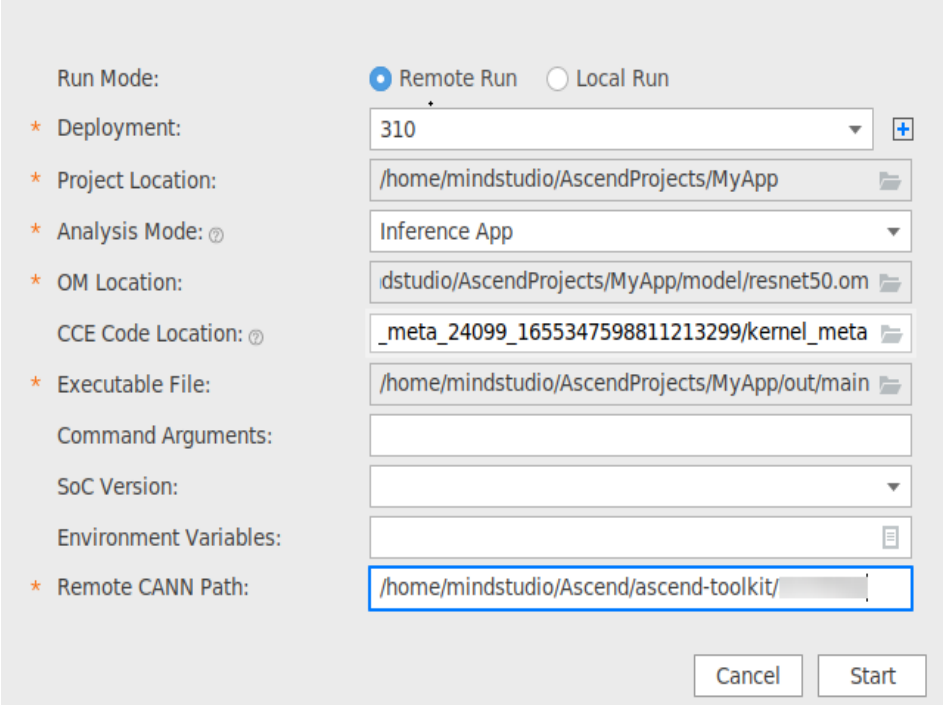


图 11-6 OM only（Windows）

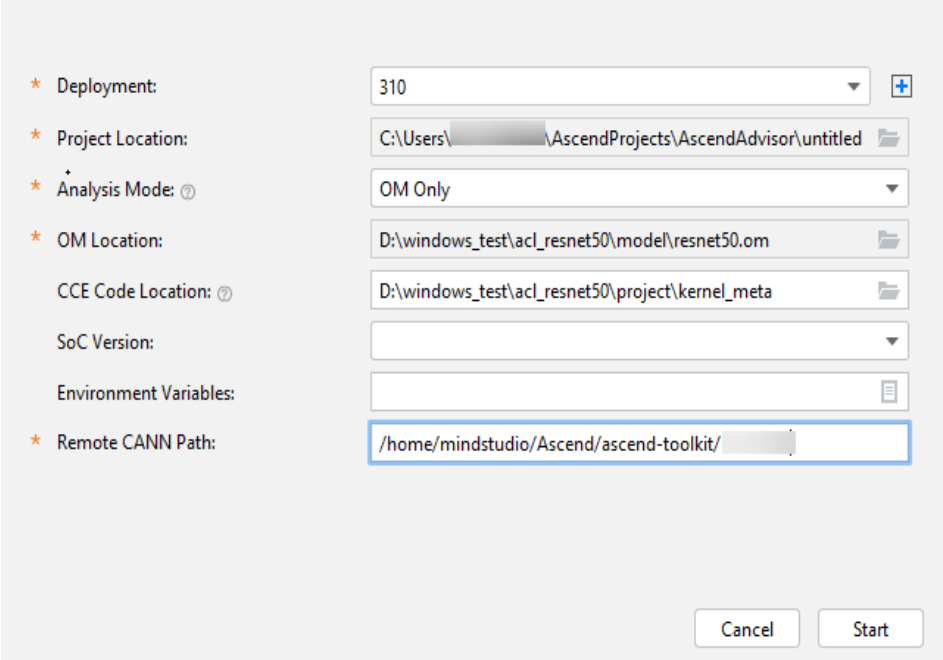
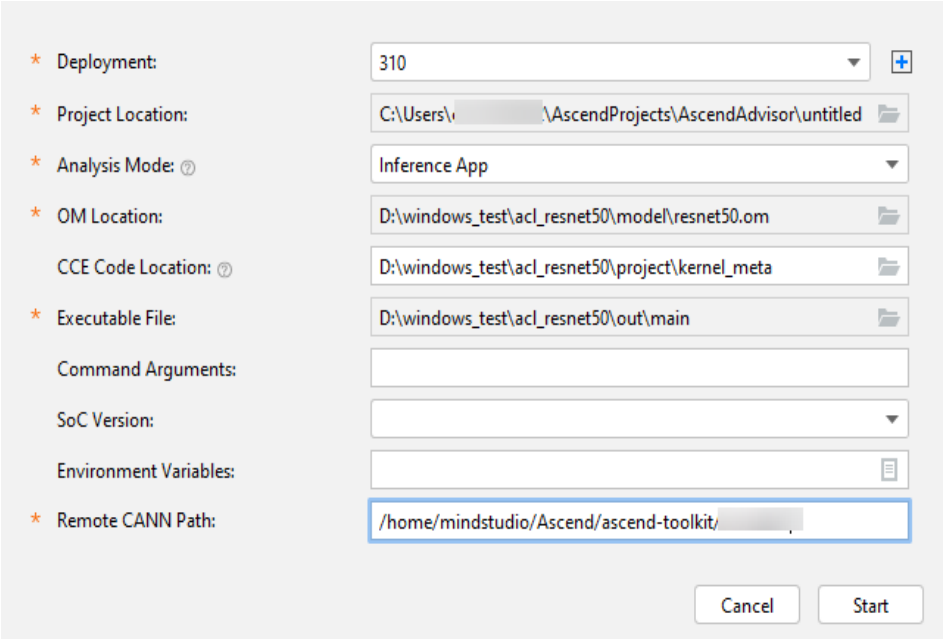


图 11-7 Inference App（Windows）



说明

- 配置完成后，Deployment和对应的Environment Variables、Remote CANN Path参数为绑定关系，单击“Start”后参数值将被保存。再次配置时，如连接已配置过的Deployment，则Environment Variables、Remote CANN Path参数自动填充，可手动修改。
- 配置13.3 Ascend Deployment映射路径关系（Mappings）时，须指定真实地址，否则无法启动专家系统分析，界面提示“The specified Deployment Mappings Local Path does not exist.”。

图 11-8 Local Run OM only

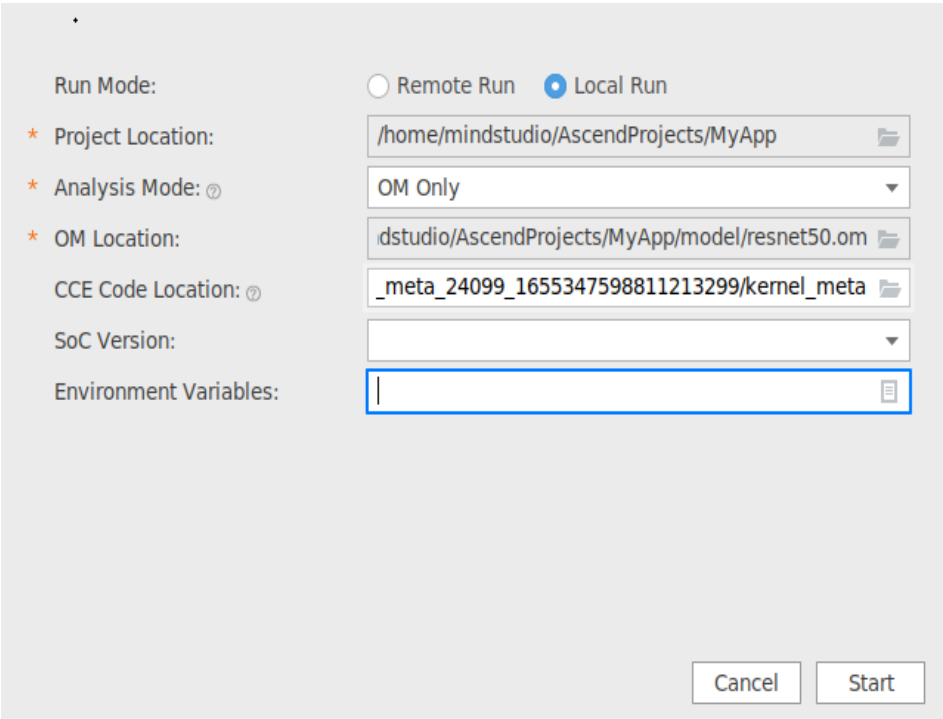
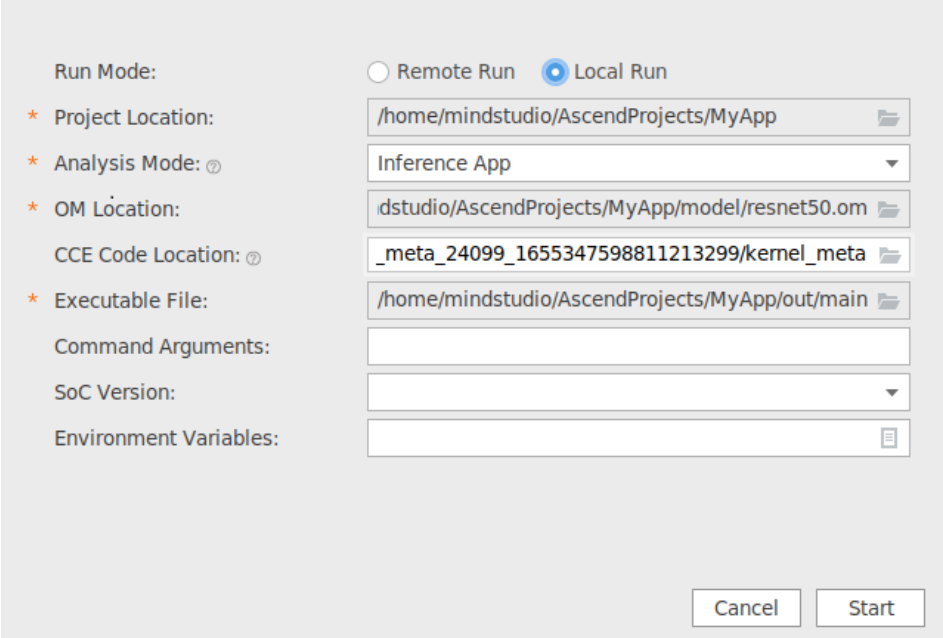


图 11-9 Local Run Inference App



说明

- OM only模式下无需指定可执行文件，直接进行内置标准专家系统分析，但分析结果可能不能完全体现用户应用的实际情况；Inference App模式需指定**Executable File**参数的可执行文件（支持指定二进制脚本文件main或Python脚本文件），可针对用户应用下的模型进行精准分析，结果更准确，但需要用户预先完成应用开发。请根据实际需求选择相应模式。
- 不建议调用与当前用户不一致的其它用户目录下的可执行文件（Executable File）和OM模型文件，避免提权风险。当选择其他用户下的文件时，系统将提示风险。
- Windows环境仅支持Remote Run模式。

表 11-4 Executable Properties 参数说明

参数	说明
Run Mode	• Remote Run：远程运行。 • Local Run：本地运行。 Windows使用场景下仅支持Remote Run，该参数不展示。
Deployment	运行配置，选择Remote Run模式时可见。通过Deployment功能，详细请参见**13.3 Ascend Deployment**，可以将指定项目中的文件、文件夹同步到远程指定机器的指定目录。
Project Location	分析完成后的.json文件保存目录，默认为${HOME}/AscendProjects/AscendAdvisor/untitled，可自定义。

参数	说明
Analysis Model	执行分析模式，取值为： • OM only：只对所选的OM模型进行分析。该模式下的可执行文件为内置标准可执行文件，无需手动指定。 • Inference App：对当前用户的应用下执行OM模型的分析，需指定**Executable File**参数的可执行文件main或Python脚本文件。 OM only模式只对模型进行标准化分析，若需要对用户应用下的模型进行精准分析请指定Inference App模式。
OM Location	指定OM模型文件，获取方式参见**11.3.2 分析功能与输入数据的对应关系**。
CCE Code Location	指定CCE文件保存目录，获取方式参见**11.3.2 分析功能与输入数据的对应关系**。不指定本参数时，专家系统的分析结果精度可能有所偏差。当前仅基于Roofline模型的算子瓶颈识别与优化建议功能需要。
Executable File	执行专家系统目标工程的可执行文件目录，支持指定二进制脚本文件main和Python脚本文件，获取方式请参见**7.3.6.1 Linux场景编译运行**，仅在指定Inference App模式时展示。须保证Remote Run模式下指定的可执行文件能够在远端环境下正常编译与执行。
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。Analysis Model选择Inference App时可见。
SoC Version	设置待分析文件所属设备的芯片版本。配置示例：Ascend*xxx*。 请根据输入数据所属环境选择对应的芯片类型，所属环境可通过**npu-smi info**命令查看芯片类型。
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。
Remote CANN Path	远端运行环境CANN软件包安装路径，选择Remote Run模式时可见。例如配置为*Ascend-cann-toolkit安装目录*/ascend-toolkit/*{version}*。

说明

- 专家系统分析过程较为复杂，会产生较多的临时文件，因此执行专家系统分析前，需预留足够大的内存空间，否则可能造成进程终止。须进入服务器查看专家系统数据目录**${data_path}/data/project**所占空间大小，例如数据目录大小为100MB时，则需要预留1GB的内存空间。
- 分析数据文件：算子仿真文件、Profiling数据文件和CCE文件的每个文件大小均不能超过100MB；OM离线模型文件大小不能超过1GB。

步骤4 完成配置后单击Start启动分析。完成分析后，展示分析结果如**11.4.1.3 分析结果展示**。

----结束

11.4.1.3 分析结果展示

说明

专家系统工具分析结果根据11.3.1 输入数据获取中所准备的文件进行分析，一次分析输出所有功能的结果，所以如果准备文件路径下未保存对应文件，则对应功能输出结果为空。

概览页

图 11-10 分析结果 Summary 页面（Model Performance Report）

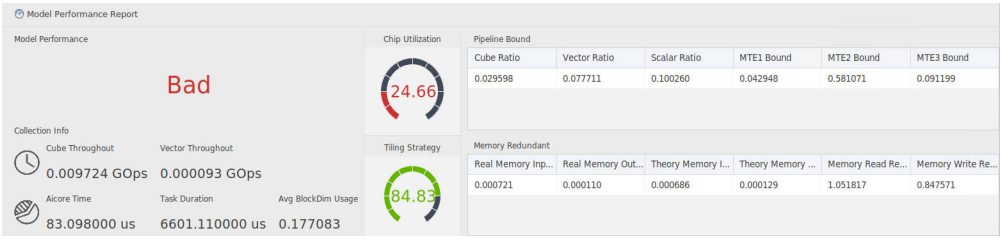


表 11-5 Model Performance Report 字段说明

字段名	字段解释
Model Performance Report	模型性能分析报告。
Model Performance	性能优劣，取值为Good/Bad。根据总体性能数据汇总计算得出。
Collection Info	汇总信息。
Cube Throughput	Cube吞吐量，单位为GOps。
Vector Throughput	Vector吞吐量，单位为GOps。
Aicore Time	AI Core执行时间，单位为us。
Task Duration	任务执行时间，单位为us。
Avg BlockDim Usage	平均BlockDim利用率，算子执行时的平均核心数，反映芯片利用情况。
Chip Utilization	芯片利用率。数值达到80为优，显示为绿色；小于80则为差，显示为红色。根据Pipeline Bound的数值计算得出。
Pipeline Bound	流水利用率。
Cube Ratio	Cube利用率。
Vector Ratio	Vector利用率。
Scalar Ratio	Scalar利用率。
MTE1 Bound	MTE1瓶颈。
MTE2 Bound	MTE2瓶颈。
MTE3 Bound	MTE3瓶颈。

字段名	字段解释
Tiling Strategy	内存读入量的数据切片策略。数值达到80为优，显示为绿色；小于80则为差，显示为红色。根据Memory Redundant的数值计算得出。
Memory Redundant	内存冗余量。
Real Memory Input(GB)	真实内存读入量，单位为GB。
Real Memory Output(GB)	真实内存写出量，单位为GB。
Theory Memory Input(GB)	理论内存读入量，单位为GB。
Theory Memory Output(GB)	理论内存写出量，单位为GB。
Memory Read Redundant	内存读入冗余系数。真实内存读入量/理论内存读入量。
Memory Write Redundant	内存写出冗余系数。真实内存写出量/理论内存写出量。

图 11-11 分析结果 Summary 页面（Computational Graph Optimization_UB Fusion）

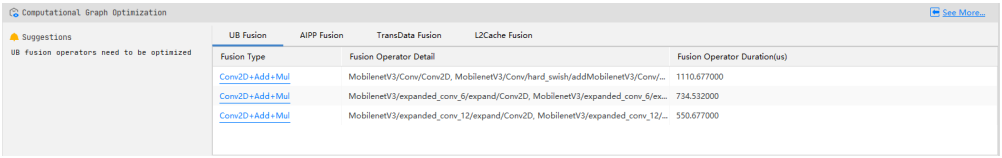


表 11-6 Computational Graph Optimization_UB Fusion 字段说明

字段名	字段解释
Computational Graph Optimization	计算图优化。算子融合推荐功能专家系统分析建议。分行展示可融合的算子。可单击栏目右上角“See More...”查看具体的可融合算子。
UB fusion operators need to be optimized	需要进行UB融合的算子。
UB Fusion	UB融合推荐。栏目下方展示可融合算子。
Fusion Type	可融合算子类型。
Fusion Operator Detail	可融合算子详细信息，算子名称之间以逗号隔开。
Fusion Operator Duration(us)	可融合算子的执行时间。单位为us。

图 11-12 分析结果 Summary 页面（Computational Graph Optimization_AIPP Fusion）

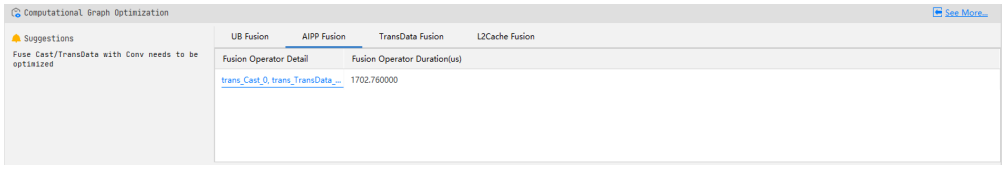


表 11-7 Computational Graph Optimization_AIPP Fusion 字段说明

字段名	字段解释
Fuse Cast/TransData with Conv needs to be optimized	需要进行AIPP首层算子融合的算子。
AIPP Fusion	AIPP融合推荐。栏目下方展示可融合算子。
Fusion Operator Detail	可融合算子详细信息，算子名称之间以逗号隔开。
Fusion Operator Duration(us)	可融合算子的执行时间。单位为us。

图 11-13 分析结果 Summary 页面（Computational Graph Optimization_TransData Fusion）

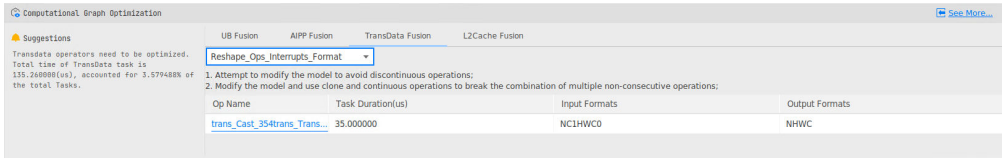


表 11-8 Computational Graph Optimization_TransData Fusion 字段说明

字段名	字段解释
TransData fusion operators need to be optimized. Total time of TransData task is *xx*(us), accounted for *xx*% of the total task.	TransData算子需要优化。 TransData算子总时长*xx*(us)，占所有任务时间占比*xx*%。
TransData Fusion	推荐消除的TransData算子。栏目下方展示可消除算子信息。
Reshape_Ops_Interrupts_Format	Reshape_Ops_Interrupts_Format的优化建议。
Attempt to modify the model to avoid discontinuous operations	在不影响精度的情况下尽量避免非连续的操作。

字段名	字段解释
Modify the model and use clone and continuous operations to break the combination of multiple non-consecutive operations	使用clone、contiguous将多个非连续操作的组合断开。
Other_Transform	Other_Transform的优化建议。
It is a reasonable scenario that transdata operation exists, for example, 4D to 5D before Conv2D.	TransData合理存在的场景，比如在Conv2D算子之前，需要把格式由4D转为5D。
Op Name	算子名称。
Task Duration(us)	算子执行持续时间。
Input Formats	算子输入格式。
Output Formats	算子输出格式。

图 11-14 分析结果 Summary 页面（Computational Graph Optimization_L2Cache Fusion）

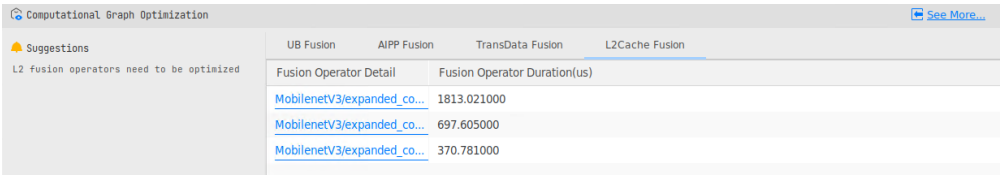


表 11-9 Computational Graph Optimization_L2Cache Fusion 字段说明

字段名	字段解释
L2 fusion operators need to be optimized	L2融合算子需要优化。
L2Cache Fusion	L2Cache融合推荐。栏目下方展示可融合算子。
Fusion Operator Detail	可融合算子详细信息，算子名称之间以逗号隔开。
Fusion Operator Duration(us)	可融合算子的执行时间。单位为us。

图 11-15 分析结果 Summary 页面（Roofline）

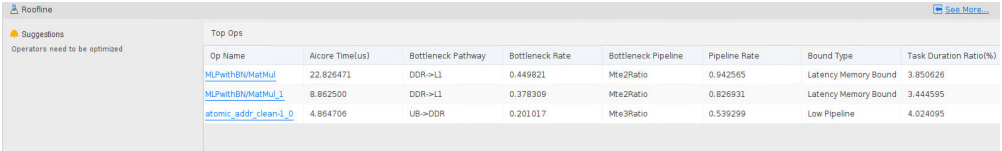


表 11-10 Roofline 字段说明

字段名	字段解释
Roofline	基于Roofline模型的算子瓶颈识别与优化建议Top3算子信息。可单击栏目右上角See More...”查看详细结果信息。
Top Ops	前三个算子。栏目下方展示Roofline模型的前三个可优化算子基本信息。
Op Name	算子名称。
Aicore Time(us)	AI Core运行时间，单位为us。
Bottleneck pathway	瓶颈通路，即工作点最靠近roofline的通路。
Bottleneck Rate	瓶颈率，即工作点占roofline上限的百分比。
Bottleneck Pipeline	占比最高的流水。
Pipeline Rate	流水最高占比。
Bound Type	瓶颈分类。
Task Duration Ratio(%)	Task耗时占比。

图 11-16 分析结果 Summary 页面（Model Graph Optimization）

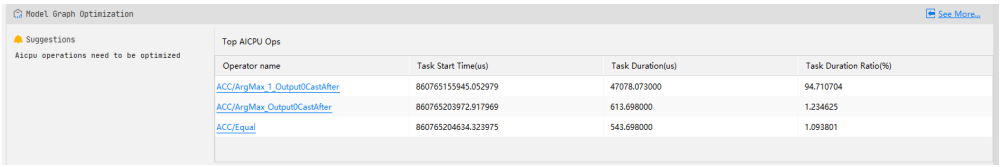


表 11-11 Model Graph Optimization 字段说明

字段名	字段解释
Model Graph Optimization	模型优化建议。
Top AICPU Ops	算子列表（按耗时从大到小排序）。
Operator name	算子名。
Task Start Time(us)	任务开始时间。
Task Duration(us)	Task耗时。
Task Duration Ratio(%)	Task耗时占比。

图 11-17 分析结果 Summary 页面（Operating Environment）

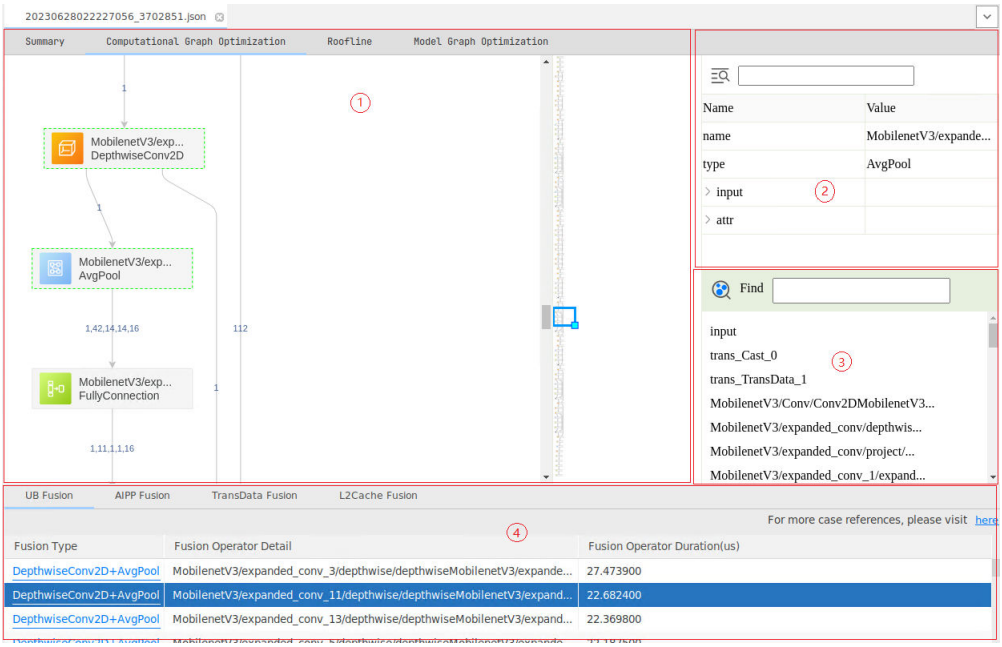
表 11-12 Operating Environment 字段说明

字段名	字段解释
Operating Environment	操作环境。
Host Operating System	Host侧操作系统。
Host Computer Name	Host侧计算机名称。
CPU Name	CPU名称。
CPU Name Type	CPU名称类型。
Control CPU Type	CtrlCPU类型。
Control CPU Number	CtrlCPU数量。
TS CPU Number	TS CPU数量。
AI CPU Number	AI CPU数量。

Computational Graph Optimization 页面（算子融合推荐功能输出结果）

Computational Graph Optimization页面输出结果包含**算子融合推荐**和**TransData算子识别**两个功能，本节介绍**算子融合推荐**功能。

图 11-18 分析结果 Computational Graph Optimization（算子融合推荐功能）



上图中各区域展示信息如下：

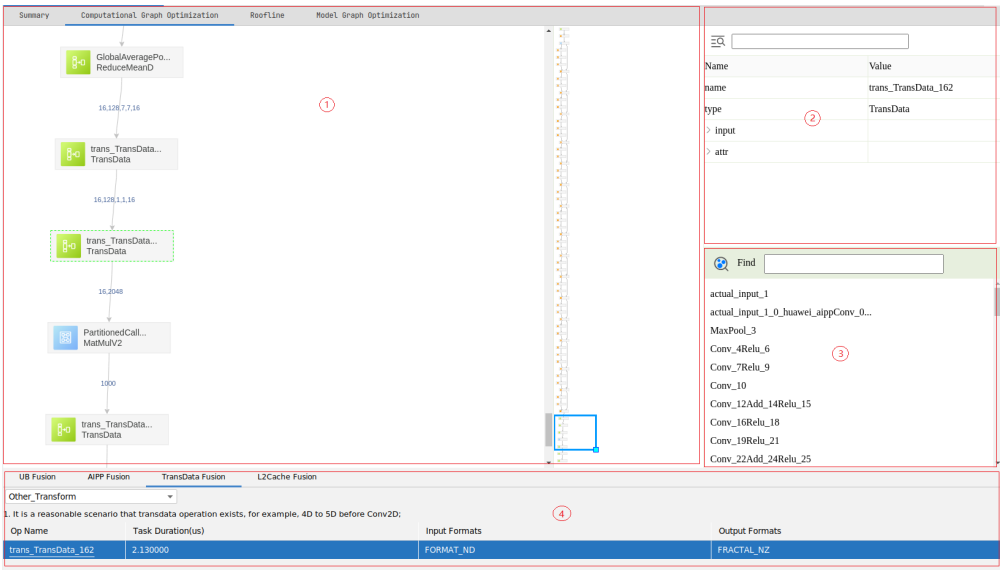
- **1区域**：展示模型可视化界面，展示模型中所有算子并高亮可融合算子。
- **2区域**：展示算子详细信息，单击可视化界面中某个算子时展示该算子信息。

- **3区域**：展示算子搜索栏，以算子名称形式显示模型中所有算子，可以搜索具体的算子名称，单击搜索栏下方算子名称也可展示对应算子详细信息。
- **4区域**：展示分析结果。对应字段解释请参见**表11-6**、**表11-7**和**表11-9**。
  - 页签部分展示**算子融合推荐**的UB、AIPP、L2Cache融合功能分析结果页签，单击可跳转到对应功能的分析结果。
  - 列表部分展示可融合算子，一行展示一条可融合信息，各个算子名以逗号隔开，内容过长的可将光标悬浮在对应行上方即可展示详细可融合算子，单击对应的行则在可视化视图上会跳转到对应算子位置。

Computational Graph Optimization 页面（TransData 算子识别功能输出结果）

Computational Graph Optimization页面输出结果包含**算子融合推荐**和**TransData算子识别**两个功能，本节介绍**TransData算子识别**功能。

图 11-19 分析结果 Computational Graph Optimization（TransData 算子识别功能）

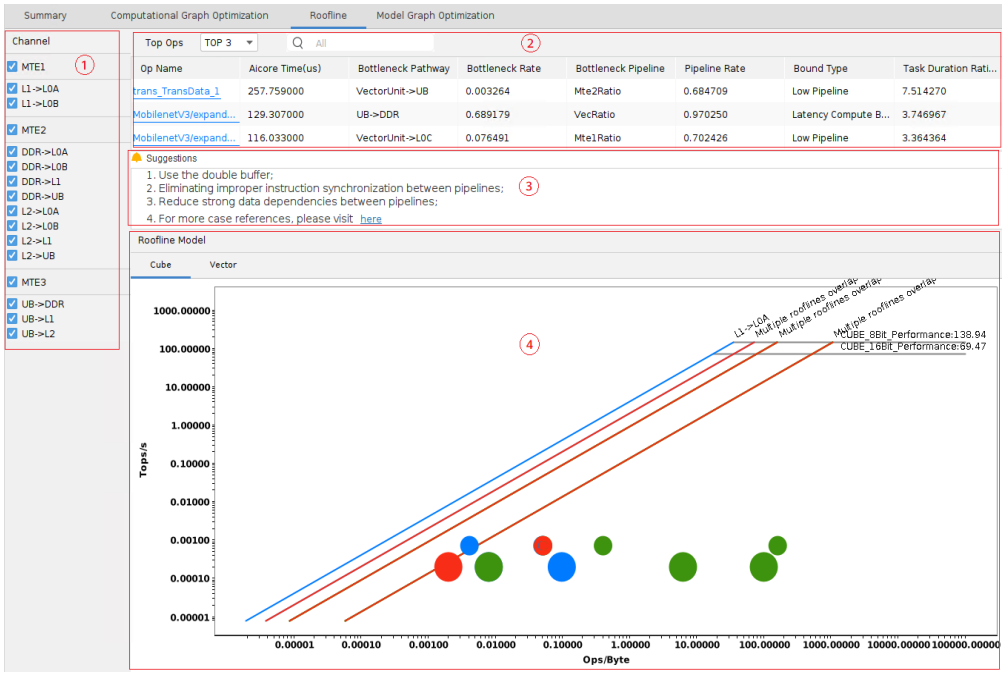


上图中各区域展示信息如下：

- **1区域**：展示模型可视化界面，展示模型中所有算子并高亮可消除算子。
- **2区域**：展示算子详细信息，单击可视化界面中某个算子时展示该算子信息。
- **3区域**：展示算子搜索栏，以算子名称形式显示模型中所有算子，可以搜索具体的算子名称，单击搜索栏下方算子名称也可展示对应算子详细信息。
- **4区域**：展示分析结果。对应字段解释请参见**表11-8**。
  - 页签部分单击TransData Fusion展示**TransData算子识别**分析结果页签。
  - 下拉框部分为不同维度的优化建议。
  - 列表部分展示可消除算子，一行展示一条可融合信息，单击对应的行则在可视化视图上会跳转到对应算子位置。

Roofline 页面（基于 Roofline 模型的算子瓶颈识别与优化建议功能输出结果）

图 11-20 分析结果 Roofline



上图中各区域展示信息如下：

- **1区域**：展示专家系统分析结果Roofline模型的Channel通路。
  - **1区域**每一项对应**4区域**中某个工作点信息，勾选表示在**4区域**中展示，去勾选表示不展示，默认全部勾选。
  - **1区域**的Channel通路与**4区域**工作点及线条颜色对应关系为：
    - Cube：MTE1-蓝色；MTE2-绿色；MTE3-红色。
    - Vector：MTE2-绿色；MTE3-红色；PIPE_V-黄色。
- **2区域**：展示TopN算子信息，可选择展示Top3、Top5和Top10，支持搜索某个算子。字段解释请参见**表11-10**。单击具体算子名可以在3区域展示对该算子专家系统建议，**4区域**显示该算子对应的工作点及瓶颈信息。
- **3区域**：展示具体存在瓶颈算子的专家系统建议，需要单击**2区域**的具体算子名。

  说明

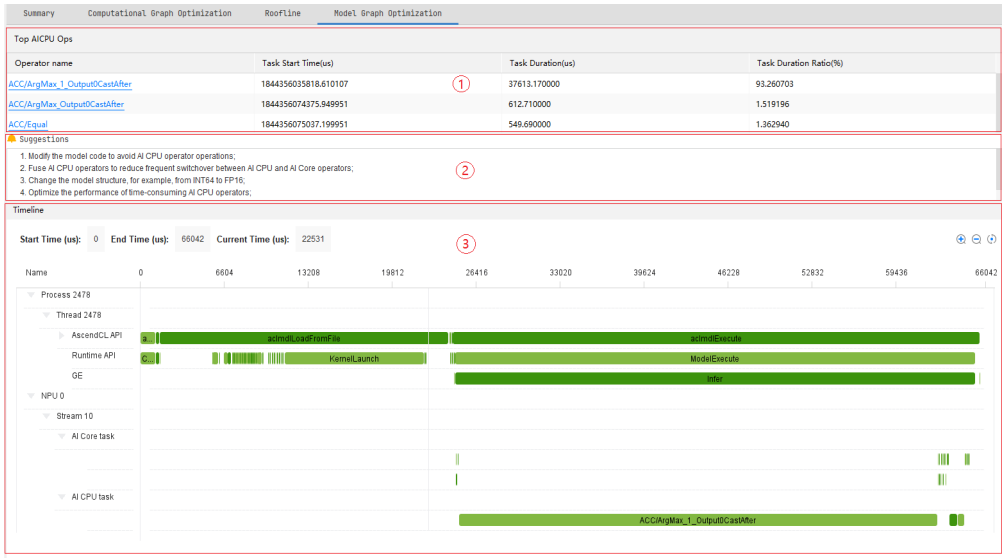
  图11-20中第4点建议为“For more case references, please visit here.”，提示有关基于Roofline模型的算子瓶颈识别与优化建议功能输出结果分析可单击链接访问**11.5.3 Roofline模型的优化分析样例**。若在结果界面上单击超链接，需要预先在Linux服务器上安装FireFox火狐浏览器。
- **4区域**：展示以坐标轴方式Roofline模型专家系统分析结果。展示算子分别在Cube和Vector计算单元下的算力情况以及理论算力和带宽。
  - 坐标轴中的横坐标单位是Ops/Byte，表示计算强度，每搬运1byte数据可以进行多少次运算，越大表示内存搬运利用率越高。
  - 纵坐标单位是Tops/s，表示运算速度，越大表示运算越快。
  - 线条转折部分将Roofline模型分成两个部分，斜线部分为Memory Bound（内存限制），横线部分为Compute Bound（计算限制），且实际工作点

（图中的彩色点）越远离对应颜色的斜线，表示Bound越严重，为主要瓶颈所在。

- 显示情况由**1区域**勾选情况决定，其中Multiple roofline overlap的斜线表示有多条Channel通路输出带宽一致导致线条重合，可通过去勾选**1区域**中相同输出带宽值的Channel项来确定对应斜线代表的是哪个Channel。
- 光标悬浮在工作点上方可显示对应工作点信息。
- 工作点按从大到小表示存在瓶颈问题的算子依次为Top1~Top3。

Model Graph Optimization 页面（基于 Timeline 的 AI CPU 算子优化功能输出结果）

图 11-21 分析结果 Model Graph Optimization



上图中各区域展示信息如下：

- **1区域**：展示Top3专家系统基于Timeline的AI CPU算子优化功能分析结果，AI CPU算子等待串行后的执行时间和优化建议。
- **2区域**：展示专家系统AI CPU算子优化建议。
- **3区域**：展示Profiling采集的Timeline视图。

说明

图11-21中2区域第5点建议为：“For more case references, please visit here.”，提示有关基于Timeline的AI CPU算子优化功能输出结果分析可单击链接访问**11.5.2 基于Timeline的AI CPU算子优化分析样例**。若在结果界面上单击超链接，需要预先在Linux服务器上安装FireFox火狐浏览器。

11.4.2 算子工程入口

11.4.2.1 概述

专家系统功能的配置入口包括专家系统入口和算子工程入口。

本节主要介绍算子工程入口的配置方式，该入口可执行的专家系统分析功能为：**算子优化分析**。

说明

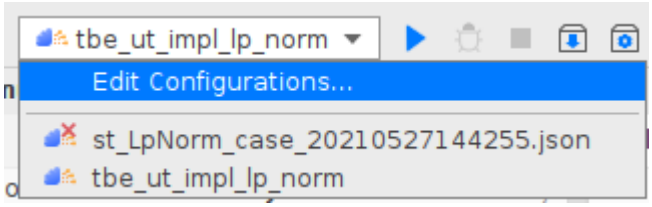
专家系统自有知识库仅提供模型或算子的可优化项并给出优化建议，具体优化方式请开发者自行修改代码。

11.4.2.2 配置步骤

步骤1 参见**8 算子开发**完成算子工程的创建和开发，并打开已完成开发的算子工程。

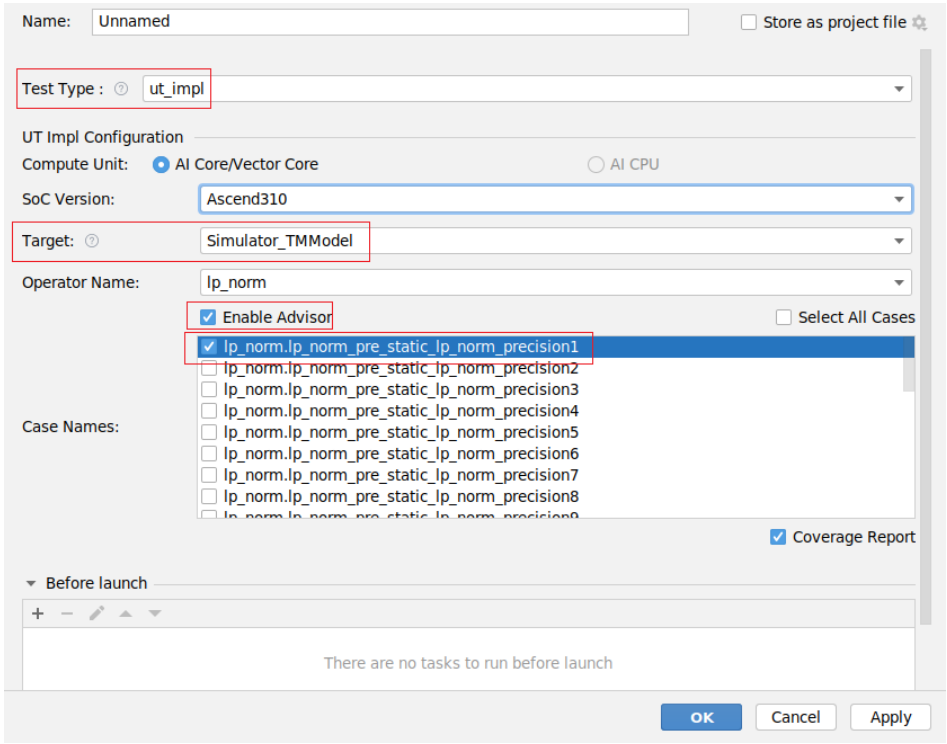
步骤2 单击算子工程界面“Run > Edit Configurations...”或单击**图11-22**所示菜单，进入运行**8.7 UT测试**配置界面。

图 11-22 快捷方式进入运行配置界面



步骤3 配置算子参数时勾选“Enable Advisor”。如**图11-23**所示。

图 11-23 运行配置界面



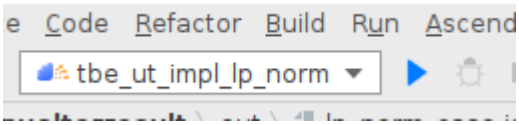
只有当Test Type参数选择ut_impl，Target参数选择Simulator_TMModel，且Case Names参数仅勾选一项Case时，才展示“Enable Advisor”开启算子专家系统功能；

当Target选择其他选项或Case Names参数勾选多个case时，“Enable Advisor”选项不显示。

步骤4 单击“OK”完成算子工程信息的配置。

步骤5 单击算子工程界面“Run >Run > ...”或单击图11-24所示菜单，运行算子UT测试。

图 11-24 快捷方式运行测试



步骤6 运行完成后专家系统的分析结果将在界面下方的日志打印窗口展示。分析结果如11.4.2.3 分析结果展示。

----结束

11.4.2.3 分析结果展示

维度一：分析向量运算

通过分析仿真dump文件：

1. 算子Vector执行效率偏低时，找出执行效率低的Vector向量指令。
2. mte2搬运粒度较低导致Vector计算的输入数据较少。
3. 是否使能多核资源。
4. 多核场景下，核之间的Vector计算量是否分配均衡。

分析结果如下：

图 11-25 分析结果展示（Vector Bound）

在Vector Bound展示结果任意一行右键可选择跳转到“TIK Code”或“CCE Code”中算子对应的位置；当算子为非TIK算子时，无“TIK Code”跳转功能；另外“CCE Code”代码跳转功能依赖于.cce和.asm文件，当工程目录（默认为out>bin>kernel_meta）下这两个文件缺少任意一个，无跳转功能。如图11-26所示。.asm文件在执行算子UT测试后自动生成，.cce文件的生成请参见设置代码跳转功能进行配置。

图 11-26 跳转选项

表 11-13 Vector Bound 输出字段参数解释

字段	说明
Vector Bound	分析向量运算。
Instruction	指令名称。
First PC	指令地址，根据取值仿真dump文件中可搜索到对应的重复向量指令。
Execution Times	向量指令被重复执行的次数。
Repeat	即repeat_times参数，表示一条向量指令的重复迭代次数，取值范围为(0,255]。
Mask	向量内参与计算的元素，取值范围为[1,128]，单位为bits。每一个bit位用来表示vector的每个元素是否参与操作，bit位的值为1表示参与计算，0表示不参与计算。

有关该算子优化分析的具体参数应用请参见《**TBE&AI CPU算子开发指南**》中的“接口参考>TBE TIK API>矢量计算>单目”章节。

优化建议：

1. Try to optimize vector instructions.
   优化Vector指令。
2. Try to optimize out->ub data move instructions.
   优化out->ub数据搬运指令。
3. Try to use multi-core resources.
   使能多核。
4. Try to reallocate vector calculations to these cores: 0 1.
   重新分配Vector计算数据到这些核：0 1

优化Vector指令可以通过修改向量指令参数mask和repeat_times，替代重复执行的向量指令。

根据字段First PC的值从仿真dump文件中可搜索到对应的重复向量指令，如上图所示Execution Times取值为64，Repeat取值为1，Mask取值为64，则该指令被重复执行了10次，而一次执行只进行了一次迭代计算，每次只计算了该向量中的一位元素。那么为了提高该指令的执行效率，可以通过修改算子代码中的repeat_times参数值为10，最后根据向量位数以及实际需求修改mask的值，修改后该指令只需执行一次即可完成10次迭代计算并且每次计算都完成向量在场景下要求计算的所有元素，从而提升该条指令执行的效率。

维度二：分析流水打断

根据仿真dump文件，针对占比最大的流水进行分析，主要从三个维度进行：

1. 其它流水导致的流水不连续。
2. 指令入队列导致的流水不连续。
3. pipe_barrier(PIPE_ALL)导致的流水打断。

根据分析的结果，对上述三个维度造成影响的周期数进行排序，结果展示如下：

图 11-27 分析结果展示（Pipeline Interruption）

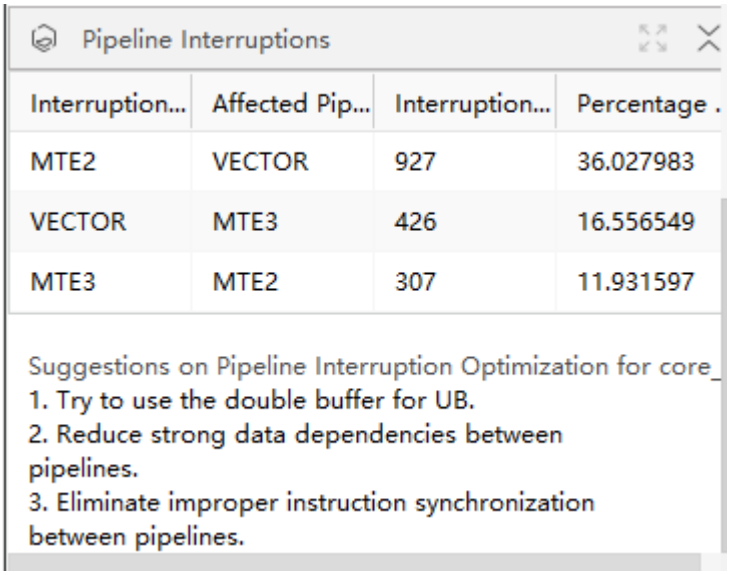


表 11-14 Pipeline Interruptions 输出字段参数解释

字段	说明
Pipeline Interruption	分析流水打断。
Interruption Factor	流水打断因素。
Affected Pipeline	受影响的流水。
Interruption Cycles	流水打断的周期数。
Percentage to Total	打断周期数占总周期数的百分比。

优化建议：

1. Try to use the double buffer for UB.
   使用乒乓策略。

2. Reduce strong data dependencies between pipelines.
   优化不合理的流水依赖。
3. Eliminate improper instruction synchronization between pipelines.
   消除流水间不合理的指令同步。
4. Delete redundant pipe_barrier(PIPE_ALL).
   删除冗余pipe_barrier（PIPE_ALL）指令。

维度三：分析标量运算

根据仿真dump文件，统计Scalar标量指令执行次数和总的执行周期，按总的执行周期的大小进行排序，选出Top5个Scalar标量指令，分析结果如下：

图 11-28 分析结果展示（Scalar Bound）

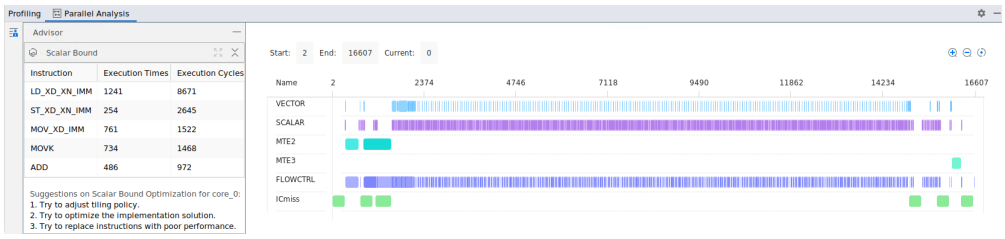


表 11-15 Scalar Bound 输出字段参数解释

字段	说明
Scalar Bound	分析标量运算。
Instruction	指令名称。
Execution Times	标量指令被重复执行的次数。
Execution Cycles	标量指令的总执行周期。

优化建议：

1. Try to adjust tiling policy.
   调整tiling策略。
2. Try to optimize the implementation solution.
   优化实现方案。
3. Try to replace instructions with poor performance.
   替换性能差的指令。

维度四：Memory bound

通过分析仿真dump文件，找出内存搬运类性能瓶颈：

1. 小包搬运：在算子一次运算内若没有达到OUT->UB/UB->OUT/L1->UB通道阈值，则判定为小包搬运。

OUT->UB/UB->OUT/L1->UB通道阈值为：Vector最大一次运算可以计算两个128长度的fp16类型向量相加或相乘，即128 * 2 * 2B = 512B。

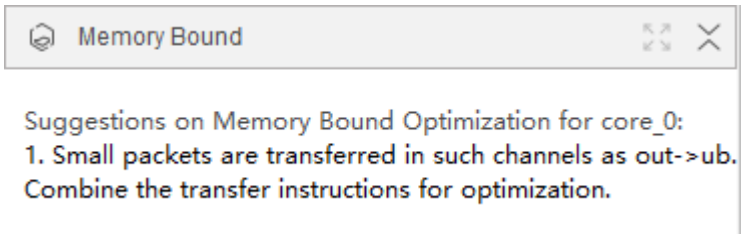
2. 冗余搬运：计算**冗余度 = 搬运量 / 计算量**，冗余度大于**1.2**则存在冗余搬运。

   搬运量为OUT->UB搬运量，**计算量**为VECTOR计算量。

3. 带宽抢占：分析OUT->UB/OUT->L1搬运指令的运行时间是否存在重合，找出可能存在的mte2带宽抢占。

分析结果如下：

图 11-29 分析结果展示（Memory bound）



优化建议：

1. Small packets are transferred in such channels as out->ub. Combine the transfer instructions for optimization.

   存在小包搬运，涉及通道OUT->UB，尝试合并搬运指令进行优化。

2. Redundant transfer exists. Optimize the data transfer policy.

   存在冗余搬运，尝试优化数据搬运策略。

3. Bandwidth preemption exists. Adjust the transfer instruction sequence.

   存在带宽抢占，尝试调整搬运指令时序。

11.5 专家系统分析样例参考

11.5.1 UB 算子融合推荐分析样例

背景介绍

算子是组成模型的基本单元。模型转换时，会进行算子融合来达到网络性能提升的目的。当模型中存在算子融合规则未覆盖且可融合的算子时，模型转换完成则无法达到最优性能。本节以YOLOV5模型为例，介绍通过专家系统UB算子融合推荐功能自动发现并输出模型中可融合的Cube和Vector算子，帮助用户快速定位可融合算子，提升模型性能。

说明

- 本文仅介绍专家系统UB算子融合推荐功能的操作及分析过程，对于OM模型中算子融合的具体操作此处不做阐述。
- 有关UB算子融合的详细介绍请参见《**图融合和UB融合规则参考**》。
- 有关模型转换的详细介绍请参见**6 模型转换和调优**。

专家系统操作

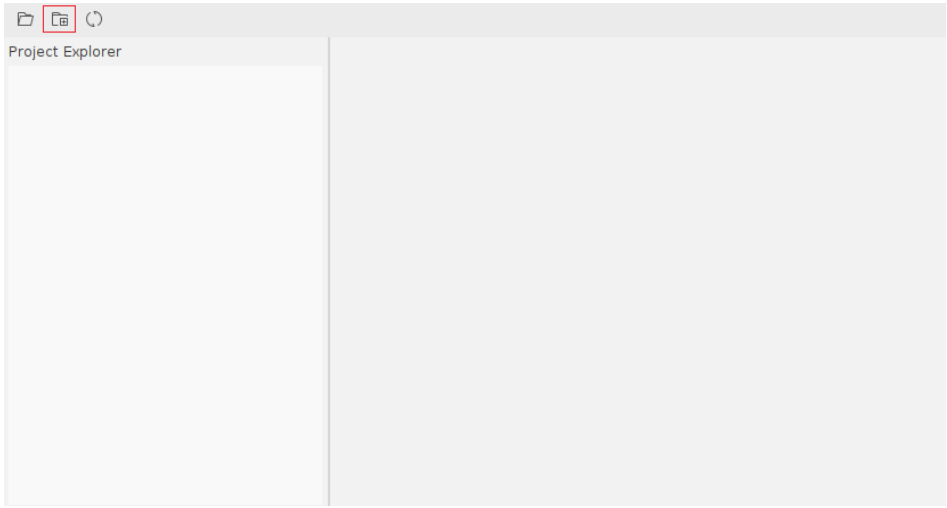
步骤1 准备通过模型转换生成的OM离线模型文件作为UB算子融合推荐功能的输入文件。

OM离线模型文件存放路径为**${data_path}**数据目录根路径的project目录下。

步骤2 单击选择并打开已编译完成的应用工程。

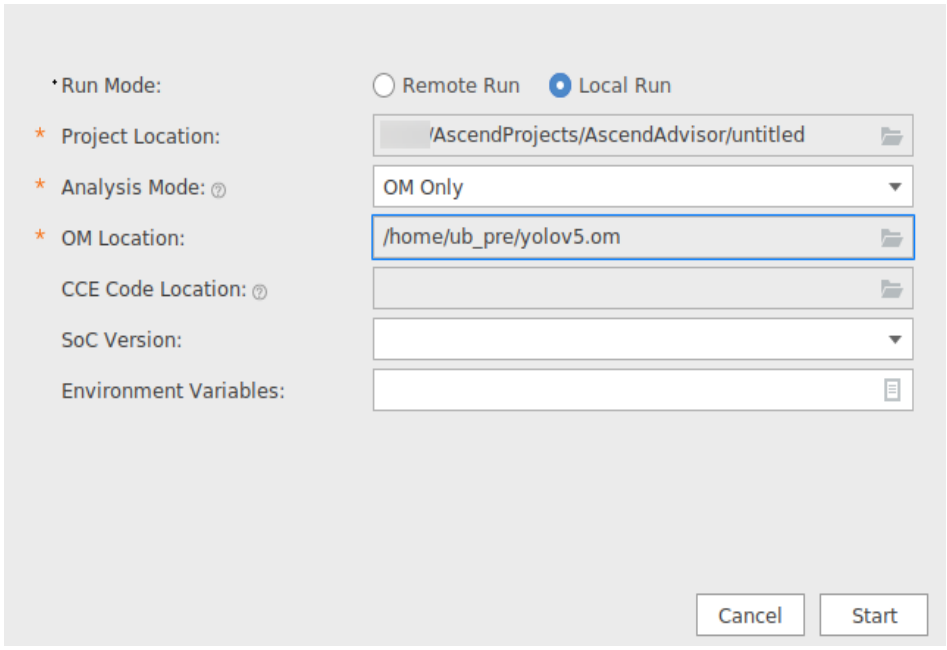
步骤3 单击菜单栏“Ascend > Advisor”，弹出专家系统工具界面 。如图11-30所示。

图 11-30 专家系统工具界面



步骤4 单击图11-30界面左上角“New Project” 按钮，打开专家系统配置界面。

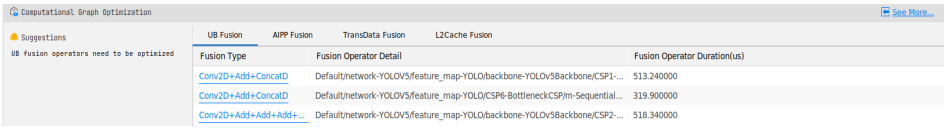
图 11-31 OM only



步骤5 如图11-31所示设置对应参数并单击Start执行分析。

步骤6 完成分析后，系统展示分析结果。输出YOLOV5模型内的可融合算子如下。

图 11-32 分析结果 Summary 页面（Computational Graph Optimization）

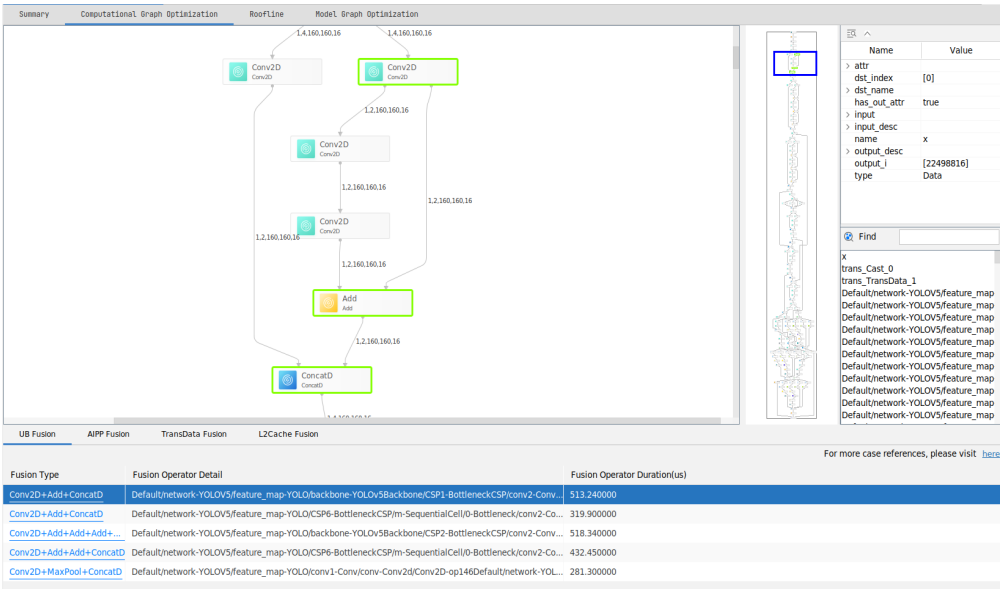


----结束

问题分析

首先查看图11-32可以大致了解可融合算子类型和名称。再通过单击See More，可以查看可融合算子在模型中的位置和明细，如图11-33所示。

图 11-33 分析结果 Computational Graph Optimization 展示



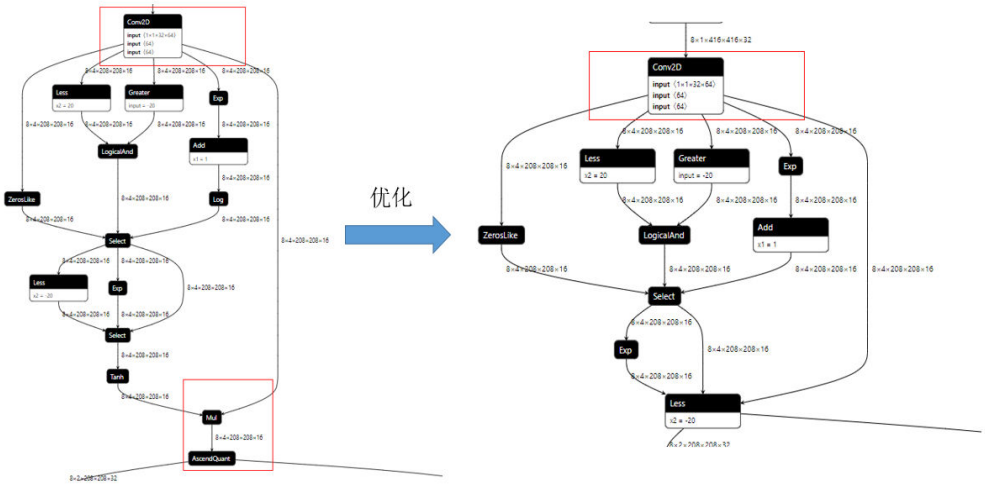
说明

由于版本更新，以上界面可能与实际情况不一致，请以实际界面为准。

问题解决

具体融合操作需开发者自行修改算子代码，完成融合后的模型结构示例如图11-34所示。

图 11-34 YOLOV5 网络模型结构（局部）



图中以YOLOV5网络模型的Conv2D、Mul和AscendQuart算子融合为Conv2D算子为例，左侧部分为融合前的YOLOV5网络模型结构，右侧部分为优化后的YOLOV5网络模型结构，可以看到优化后模型结构得到了简化。

结论

通过专家系统工具对OM模型的UB算子融合分析可知专家系统工具可以快速准确定位到模型中的可融合算子，减少用户的定位时间，提升效率。

11.5.2 基于 Timeline 的 AI CPU 算子优化分析样例

背景介绍

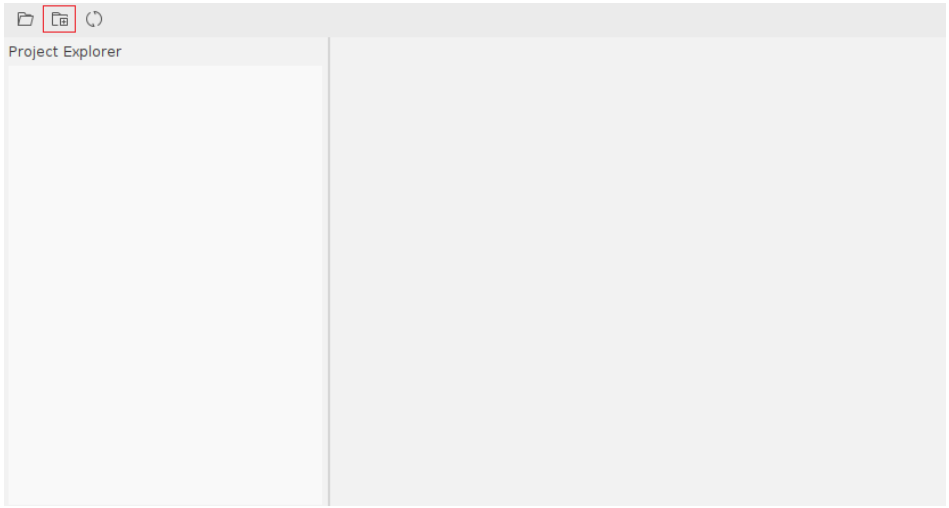
AI CPU是昇腾AI处理器计算单元，因为该CPU计算处理单位自身瓶颈，导致运行在AI CPU上的算子影响模型的执行时间。AI CPU算子的优化往往是重点关注点和优化对象。本节以MLP模型为例，介绍通过专家系统AI CPU识别功能， 自动识别串行执行AI CPU算子，给出优化建议，提升模型整体性能。

专家系统操作

步骤1 单击选择并打开已编译完成的应用工程。

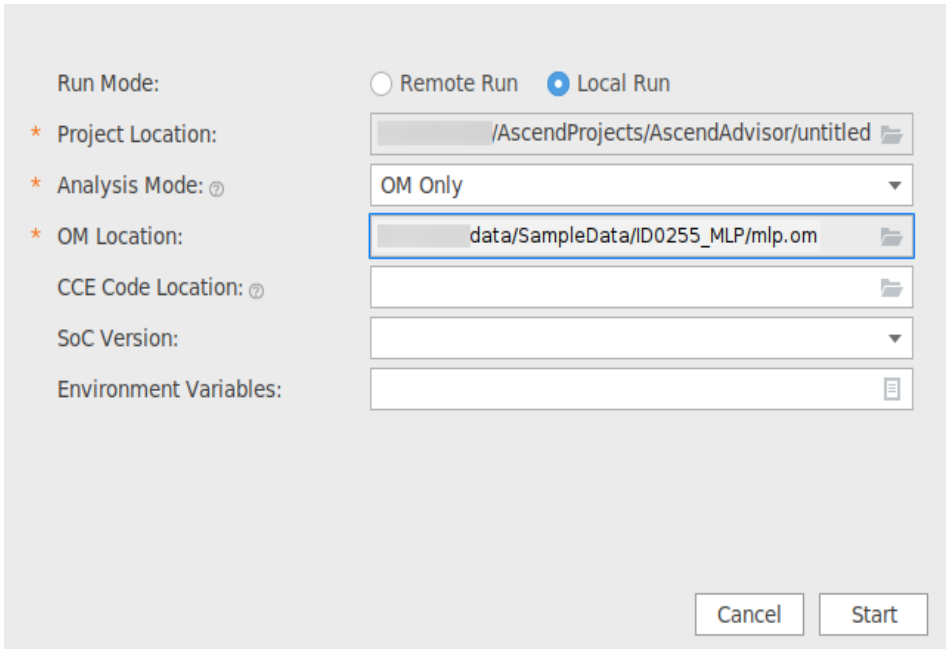
步骤2 单击菜单栏“Ascend > Advisor”，弹出专家系统工具界面 。如图11-35所示。

图 11-35 专家系统工具界面



步骤3 单击专家系统功能界面左上角“New Project” 按钮，打开专家系统配置界面。

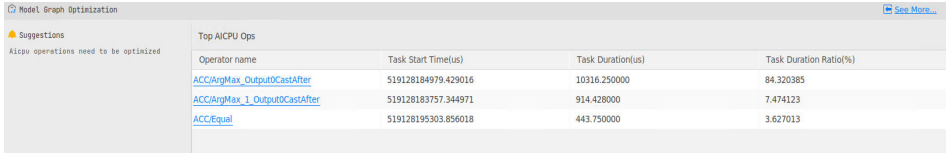
图 11-36 OM only



步骤4 如图11-36所示设置对应参数并单击Start执行分析。

步骤5 完成分析后，系统展示分析结果。输出MLP模型内的AI CPU算子如下。

图 11-37 分析结果 Summary 页面（Model Graph Optimization）



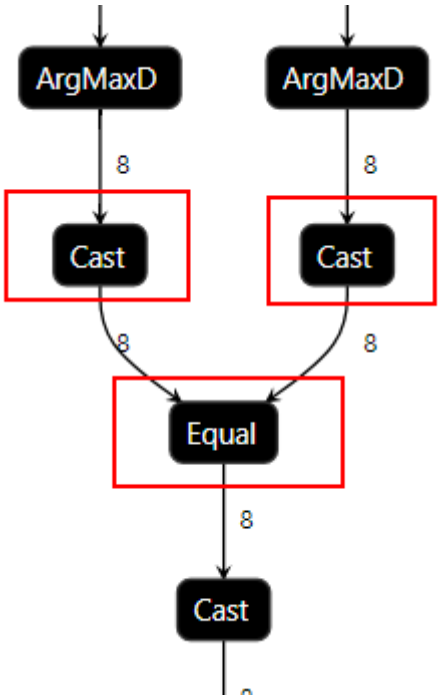
----结束

问题分析

根据专家系统AI CPU算子识别的推荐结果，发现计算图中两个Cast算子和Equal算子都在AI CPU上执行，需要考虑将Cast算子和Equal算子在AI CPU上消除。

由于Cast算子的作用是将算子输出结果进行格式转换，且在AI Core上的运算仅支持32位及以下的数据类型，根据图11-38，可推测下层的Equal算子数据类型应该在32位以上，所以生成了负责数据格式转换的Cast算子且这两个算子转移到了AI CPU上进行运算。

图 11-38 MLP 网络模型结构（局部）



问题解决

根据图11-37优化建议“3. Change the model structure, for example, from INT64 to FP16.（更改模型结构，如：INT64转FP16。）”。

可将Equal算子数据类型修改为与上层ArgMaxD算子一样的数据类型，即可消除掉Cast算子并将Equal算子转移回AI Core上运算。

修改Equal算子后再次进行专家系统分析可以发现Cast算子和Equal算子已不在AI CPU算子优化分析结果中，同时还输出UB融合推荐发现还可以对ArgMaxD和Equal算子进行融合，进一步提升效率。相关分析方式请参见11.5.1 UB算子融合推荐分析样例。

结论

通过专家系统工具的分析，可以快速找出AI CPU算子。并根据专家系统的提示，结合实际的网络模型实际特点，对问题分析后得到解决方案。提升了网络性能问题分析效率。

11.5.3 Roofline 模型的优化分析样例

背景介绍

Davinci芯片架构复杂化，人工基于Profiling数据识别性能瓶颈，门槛高、耗时长。Roofline模型以运算强度为横轴、每秒浮点运算次数为纵轴画图，描述芯片最佳性能和实际运行负载特性，反馈优缺点，给出优化方向。

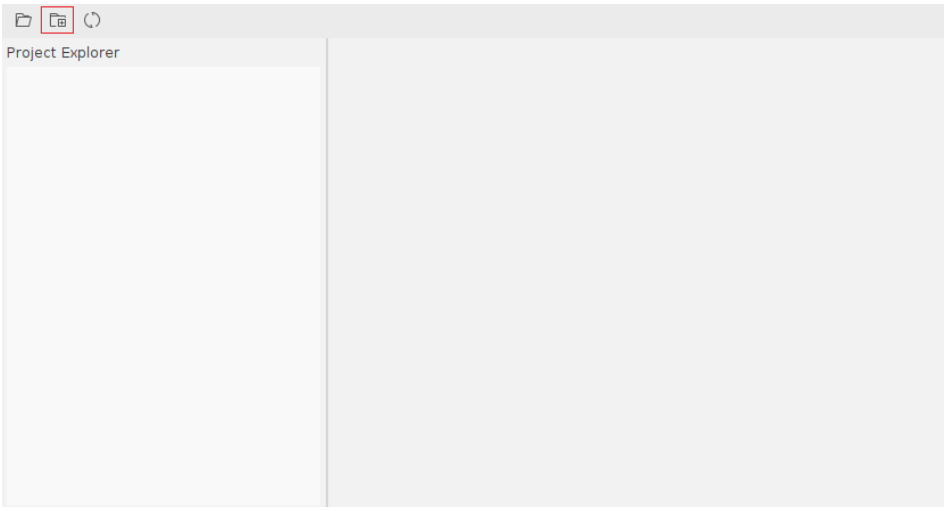
专家系统操作

步骤1 准备模型mobilenetv3，通过Profiling采集生成的profiling文件，ATC转换生成的OM文件、cce文件作为Roofline功能的输入文件。文件路径为**${data_path}**数据目录根路径。

步骤2 单击选择并打开已编译完成的应用工程。

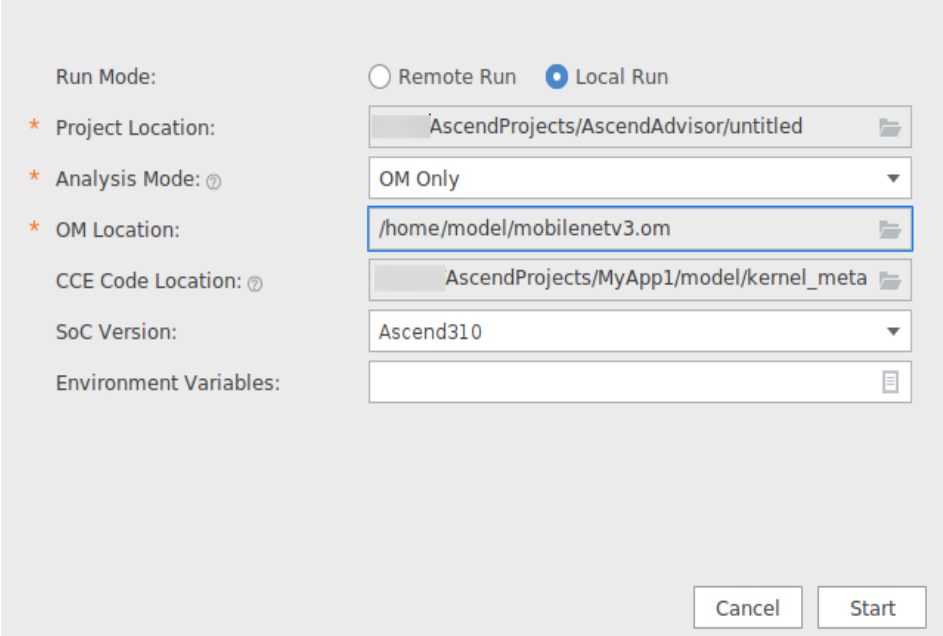
步骤3 单击菜单栏“Ascend > Advisor”，弹出专家系统工具界面 。如图11-39所示。

图 11-39 专家系统工具界面



步骤4 单击图11-39界面左上角“New Project” 按钮，打开专家系统配置界面。

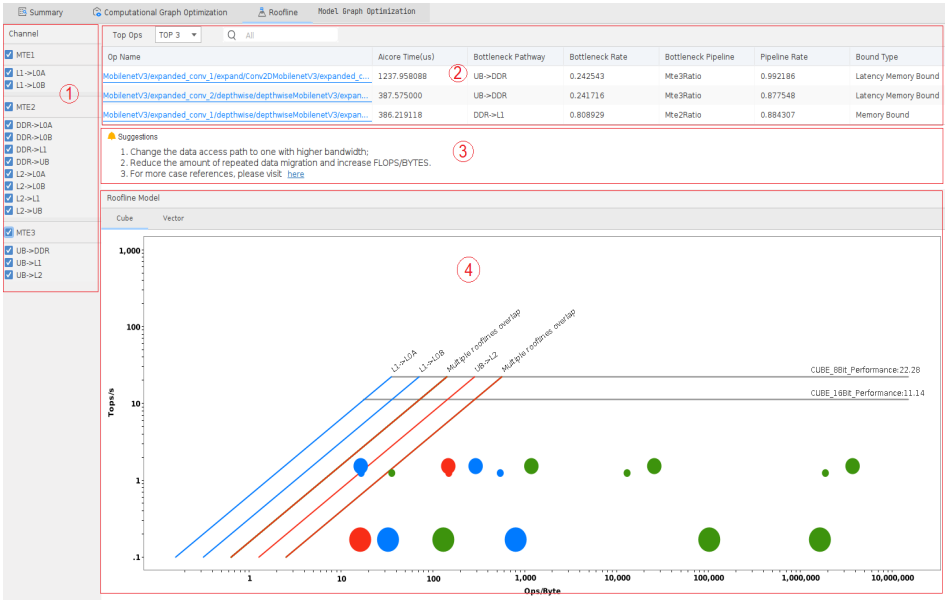
图 11-40 OM only



步骤5 如图11-40所示设置对应参数并单击Start执行分析。

步骤6 完成分析后，系统展示分析结果。输出mobilenetv3模型的Roofline分析结果如下。

图 11-41 分析结果 Roofline 展示



----结束

问题分析

1. 命令行结果展示TopN算子信息，字段解释如下。结果按照AI Core运行时间降序排列，优先解决时间占比高的算子问题。

字段	说明

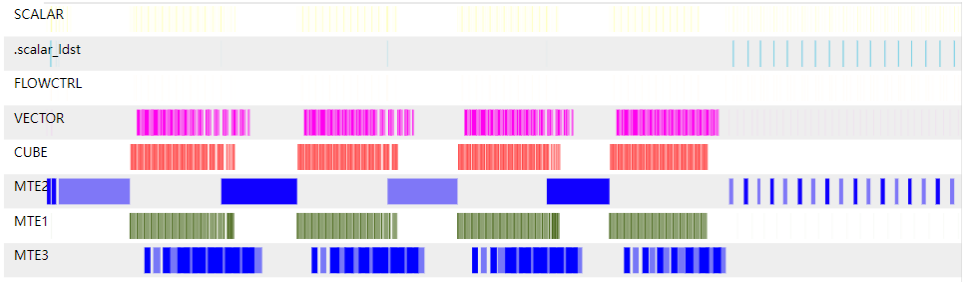
Op Name	算子名称。
AICore Time(us)	AI Core运行时间，单位为us。
Bottleneck pathway	瓶颈通路，即工作点最靠近roofline的通路。
Bottleneck Rate	瓶颈率，即工作点占roofline上限的百分比。
Bottleneck Pipeline	占比最高的流水。
Pipeline Rate	流水最高占比。
Bound Type	瓶颈分类。

2. 根据Roofline结果，可以快速获取当前模型的性能瓶颈点以及性能瓶颈通路。其中，AI Core Time与Bottleneck Rate可以综合衡量模型优化收益，如MobilenetV3/Conv/Conv2D的瓶颈率为25.91%，算子时间为293.7us，若优化后能提升到80%，收益估计为200us。
3. 以下举例说明几个算子的分析过程：
   a. 根据Roofline结果，MobilenetV3/Conv/Conv2D算子为L2->L1 latency memory bound，bound ratio=25.91%。分析该算子的流水并行度，发现该算子耗时最大的为MTE2流水，占算子总耗时的91.50%，可知该算子阻塞点在MTE2流水搬运。可以检查数据搬运粒度是否过小，或存在搬运依赖。
   b. MobilenetV3/expanded_conv/depthwise/depthwise 算子为L1->L0A latency memory bound，bound ratio=35.45%。首先分析该算子的流水并行度，发现该算子耗时最大的为MTE2流水，占算子总耗时的46.66%，低于80%, 可知流水并行存在问题。
   c. MobilenetV3/expanded_conv_3/squeeze_excite/AvgPool 算子，距离屋顶最近的数据通路为L2->UB，bound ratio=13.86%，算子属于latency bound，首先进行流水并行度 分析，发现占比最大的Vector Pipeline，占比85.11%，说明流水并行度较好。但是Vector Pipeline耗时占比最大，在Roofline里面确实latency bound说明Vector的计算过程存在问题。需要分析Vector问题。

问题解决

1. 不同的算子需要根据算子的具体问题具体制定解决策略。这一节主要介绍如何根据专家系统Roofline模型的优化建议，快速识别性能问题。
2. 例子1：分析MobilenetV3/expanded_conv/depthwise/depthwise算子的性能问题。根据专家系统的分析结果可知，该算子硬件利用率低，且各通路的瓶颈占比都很低，该算子为流水并行存在问题。根据优化建议“Reduce strong data dependencies between pipelines”，首先分析该算子各流水之间是否存在同步依赖。查看算子仿真流水图如下。MTE2流水中间存在很长一段时间的空闲时间，并且MTE2流水阻塞了MTE1 流水。确实存在流水打断的问题。

图 11-42 流水打断



3. 原始cce代码中首先使用MTE2搬运很大一部分Featuremap去进行cube计算，在cube计算结束后进行第二部分 Featuremap的搬运。

```
...
//第一次搬运Featuremap
copy_gm_to_cbuf(((__cbuf__ half *)fmap_local_L1_1), ((__gm__ half *)fmap +
(((int32_t)block_idx) * 401408)), 0, 1, 12544, 0, 0, PAD_NONE);
...
//Cube运算
mad(((__cc__ float *)mad_local_L0C2), ((__ca__ half *)im2col_fractal_local_L0A_8), ((__cb__
half *)filter_reshape_local_L0B_9), 448, 16, 16, (int8_t)0ULL);
...
//Vector运行
vadd(((__ubuf__ half *)depthwise_cast), ((__ubuf__ half *)depthwise_cast), ((__ubuf__ half
*)bias_tensor_local_UB), (uint8_t)56ULL, (uint8_t)1ULL, (uint8_t)1ULL, (uint8_t)0ULL,
(uint8_t)8ULL, (uint8_t)8ULL, (uint8_t)0ULL);
...
//第二次搬运Featuremap
copy_gm_to_cbuf(((__cbuf__ half *)fmap_local_L1_12), ((__gm__ half *)fmap +
((((int32_t)block_idx) * 401408) + 200704)), 0, 1, 12544, 0, 0, PAD_NONE);
```

因此可以调整第二次Featuremap搬运指令的位置，将第二次Featuremap的搬运放到第一次搬运后面，这样第一次搬运完成之后可以直接进行第二次搬运。减少不同指令之间的数据依赖。

4. 另外，根据优化建议“Eliminating improper instruction synchronization between pipelines”，代码中可能存在不合理的同步指令。分析CCE代码，发现代码中两条流水之间有pipe_barrier(PIPE_ALL) 指令。可以删除该指令进行优化。

结论

通过专家系统工具的分析，可以快速找出模型的Top瓶颈问题。并根据不同的瓶颈类型，给出对应的优化建议。根据优化意见，可以有针对性的进行性能分析，提升了模型调优效率。

12 算子和模型速查

12.1 简介

算子和模型速查工具用于查询当前版本CANN支持的算子信息和ModelZoo支持的模型信息。支持的算子列表是通过解析CANN中的算子包获得；支持的模型列表来源于昇腾社区**ModelZoo**。使用该功能前用户需确保已安装requests库。

12.2 查询算子

使用约束

- 当前算子速查工具支持查询Atlas 200/300/500 推理产品、Atlas 训练系列产品和Atlas 推理系列产品上的算子。
- 如果当前未安装有效的Ascend-cann-toolkit开发套件包，将不显示**步骤1**中两种入口，请检查Ascend-cann-toolkit开发套件包是否正确安装。

算子查询

步骤1 进入算子查询界面方式如下所示。

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

步骤2 弹出“Supported Operator Info”界面，如**图12-1**所示。

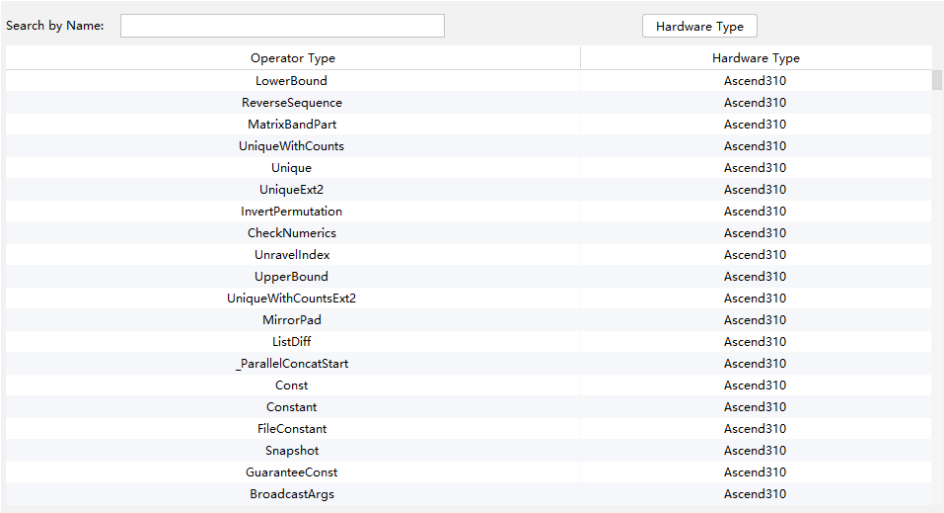
说明

查询结果请以实际情况为准，以下示例仅供参考。

- 用户可在“Search by Name”输入需要查询的算子类型名称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）

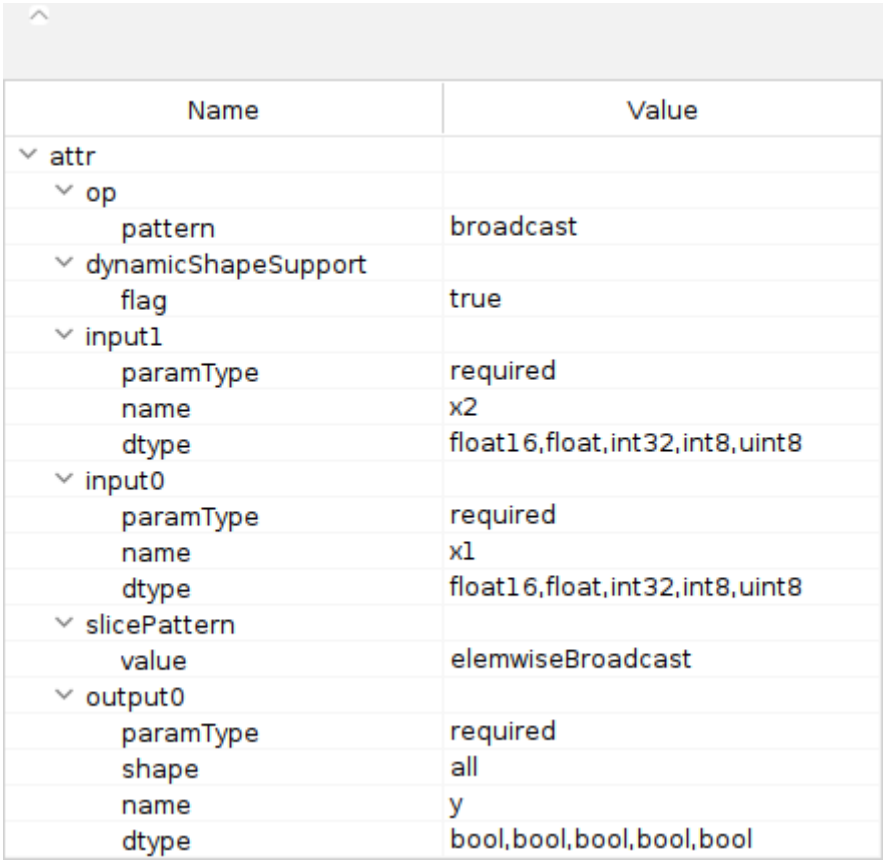
图 12-1 Supported Operator Info（示例）



Operator Type	Hardware Type
LowerBound	Ascend310
ReverseSequence	Ascend310
MatrixBandPart	Ascend310
UniqueWithCounts	Ascend310
Unique	Ascend310
UniqueExt2	Ascend310
InvertPermutation	Ascend310
CheckNumerics	Ascend310
UnravelIndex	Ascend310
UpperBound	Ascend310
UniqueWithCountsExt2	Ascend310
MirrorPad	Ascend310
ListDiff	Ascend310
_ParallelConcatStart	Ascend310
Const	Ascend310
Constant	Ascend310
FileConstant	Ascend310
Snapshot	Ascend310
GuaranteeConst	Ascend310
BroadcastArgs	Ascend310

- 选中某个算子类型，可在详情页面查看算子具体信息，如图12-2所示。

图 12-2 算子详情示例



Name	Value
attr	
op	
pattern	broadcast
dynamicShapeSupport	
flag	true
input1	
paramType	required
name	x2
dtype	float16,float,int32,int8,uint8
input0	
paramType	required
name	x1
dtype	float16,float,int32,int8,uint8
slicePattern	
value	elemwiseBroadcast
output0	
paramType	required
shape	all
name	y
dtype	bool,bool,bool,bool,bool

----结束

12.3 查询模型

使用约束

- 支持的模型信息是从昇腾开发者社区的**ModelZoo**获取。

  说明

  MindStudio后台是通过ModelZoo.ini文件中的URL：https://apigw-cn-south.huawei.com/api/ascendcommunity/modelZoos/versions使用包含X-HW-ID：com.huawei.biz.idp_public的请求头来获取昇腾开发者社区ModelZoo上支持的模型信息。

- 如果当前未安装有效的Ascend-cann-toolkit开发套件包，将不显示**步骤1**中两种入口，请检查Ascend-cann-toolkit开发套件包是否正确安装。

模型查询

步骤1 进入模型查询界面，两种进入方式如下所示。

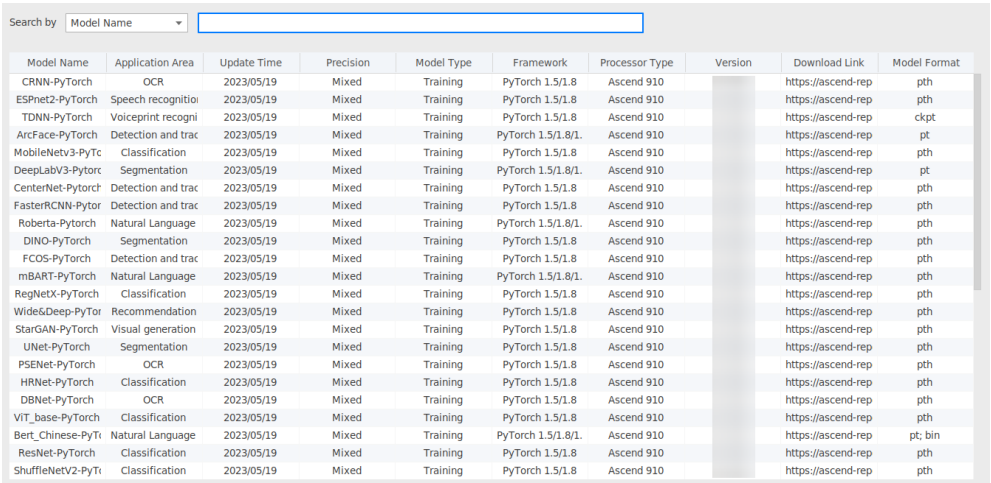
- 在工具栏单击图标  。
- 在菜单栏选择“Ascend > View Supported Models”。

步骤2 如果用户环境可以联网，成功获取后台信息，则弹出“Supported Model Info”界面，显示预训练好的模型，如**图12-3**所示。界面参数信息说明如**表12-1**所示。

说明

- 如果安装的CANN为6.0.RC1之前的版本，且界面显示中文乱码，请参见**14.3.5 中文显示乱码和界面显示不全不规整**处理。
- 查询结果请以实际情况为准，以下示例仅供参考。

图 12-3 Supported Model Info（示例）



Model Name	Application Area	Update Time	Precision	Model Type	Framework	Processor Type	Version	Download Link	Model Format
CRNN-PyTorch	OCR	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
ESPnet2-PyTorch	Speech recognitio	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
TDNN-PyTorch	Voiceprint recogni	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	ckpt
ArcFace-PyTorch	Detection and trac	2023/05/19	Mixed	Training	PyTorch 1.5/1.8/1.	Ascend 910		https://ascend-rep	pt
MobileNetv3-PyTc	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
DeepLabV3-Pytorc	Segmentation	2023/05/19	Mixed	Training	PyTorch 1.5/1.8/1.	Ascend 910		https://ascend-rep	pt
CenterNet-Pytorch	Detection and trac	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
FasterRCNN-Pytor	Detection and trac	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
Roberta-Pytorch	Natural Language	2023/05/19	Mixed	Training	PyTorch 1.5/1.8/1.	Ascend 910		https://ascend-rep	pth
DINO-PyTorch	Segmentation	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
FCOS-PyTorch	Detection and trac	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
mBART-PyTorch	Natural Language	2023/05/19	Mixed	Training	PyTorch 1.5/1.8/1.	Ascend 910		https://ascend-rep	pth
RegNetX-PyTorch	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
Wide&Deep-PyTor	Recommendation	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
StarGAN-PyTorch	Visual generation	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
UNet-PyTorch	Segmentation	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
PSENet-PyTorch	OCR	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
HRNet-PyTorch	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
DBNet-PyTorch	OCR	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
ViT_base-PyTorch	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
Bert_Chinese-PyT	Natural Language	2023/05/19	Mixed	Training	PyTorch 1.5/1.8/1.	Ascend 910		https://ascend-rep	pt; bin
ResNet-PyTorch	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth
ShuffleNetV2-PyT	Classification	2023/05/19	Mixed	Training	PyTorch 1.5/1.8	Ascend 910		https://ascend-rep	pth

表 12-1 参数说明

参数	说明
Model Name	模型名称。

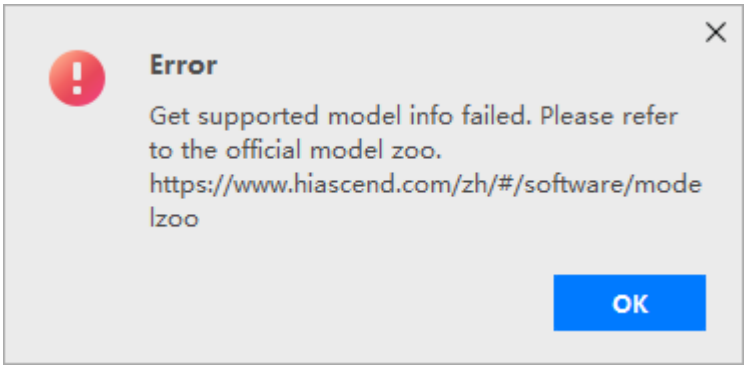
参数	说明
Application Area	应用领域，例如Image Classification、Object Detection。
Update Time	更新时间。
Precision	模型精度，例如FP16、INT8。
Model Type	模型类型，例如Offline Model。
Framework	框架，例如PyTorch、TensorFlow。
Processor Type	处理器类型。
Version	模型版本信息。
Download Link	下载链接，可以复制对应的链接粘贴到浏览器上打开，获取相应的模型文件。
Model Format	模型格式，例如om、onnx。

可以通过“Search by”后的下拉框选择支持的信息字段并在输入框中补充关键字进行模型信息的搜索和筛选。

获取后台信息失败或者获取ModelZoo的配置文件URL被修改时，请参考如下方法处理。

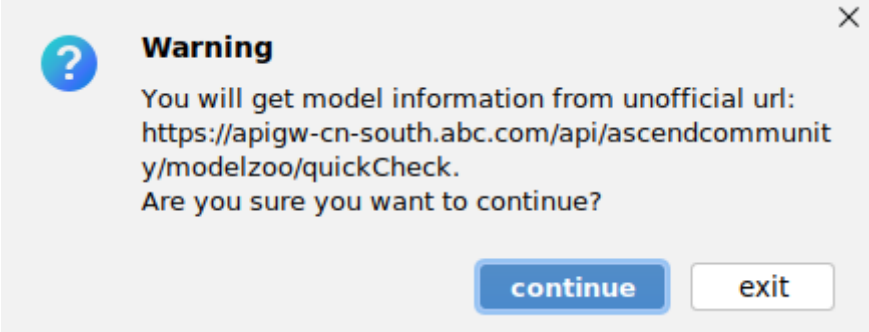
- 如果获取后台信息失败，则弹出ModelZoo对话框，显示ModelZoo的URL，如图12-4所示。用户可直接访问该URL查询支持的模型。

  图 12-4 ModelZoo

- 如果获取ModelZoo的配置文件URL被修改，使得URL域名为非.huawei.com结尾，则弹出如图12-5所示对话框。

图 12-5 警告窗口（示例）



- continue：继续通过该URL查询支持的模型。
- exit：退出查询。

----结束

13 基础操作

13.1 启动 MindStudio

linux 系统

Linux系统参见如下方式启动MindStudio：

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，执行以下命令查询进程：

```
ps -ef|grep idea
```

说明

如果启动MindStudio时，提示无法为对象堆保留足够空间，请参见**14.3.12 设置超大内存后无法启动MindStudio**处理。

Windows 系统

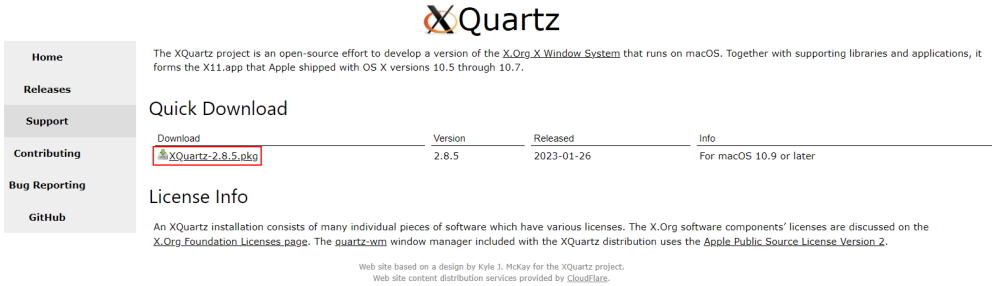
在Windows桌面，双击MindStudio应用程序启动MindStudio。

Mac 系统

Mac系统通过SSH连接启动MindStudio，具体方式如下所示：

步骤1 根据Mac系统版本下载对应的XQuartz，用于实现Linux图形化界面显示，单击**链接**下载并安装XQuartz，如**图13-1**所示。

图 13-1 下载 XQuartz

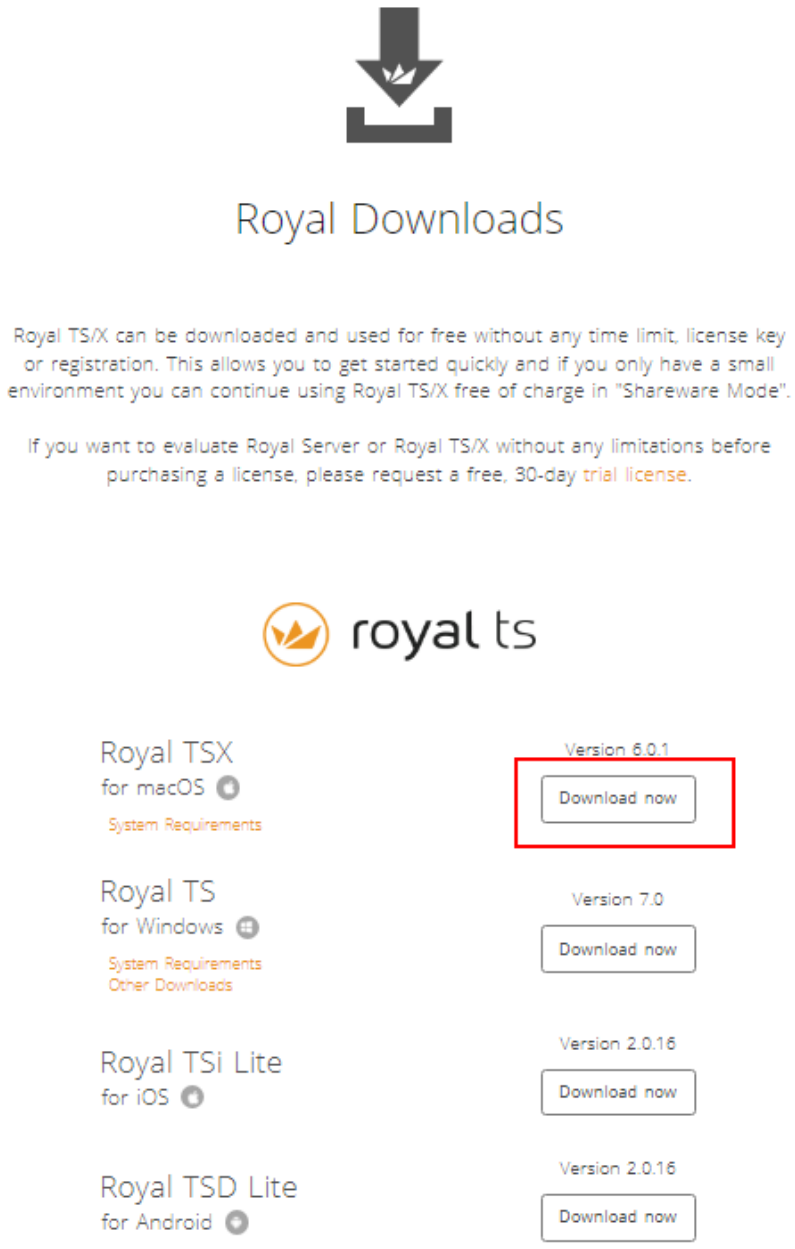


步骤2 配置环境变量，两种方式如下。

- 方式一：打开本地终端，输入如下命令。
  ```
  export DISPLAY=:0
  ```
- 方式二：在远程服务器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 DISPLAY=:0
  ```
  执行**:wq!**命令保存文件并退出。
  执行**source ~/.bashrc**命令使其立即生效。

步骤3 单击**链接**下载并安装远程桌面管理工具Royal TSX，如**图13-2**所示。

图 13-2 下载 Royal TSX



步骤4 安装完成后打开Royal TSX，单击界面上方菜单栏中的“Royal TSX > Plugins...”，如图13-3所示；在Plugins页签“Available Plugins”中搜索并安装Terminal（支持SSH连接功能）和File Transfer（支持文件传输功能）插件，如图13-4所示。

图 13-3 进入 Plugins 安装界面

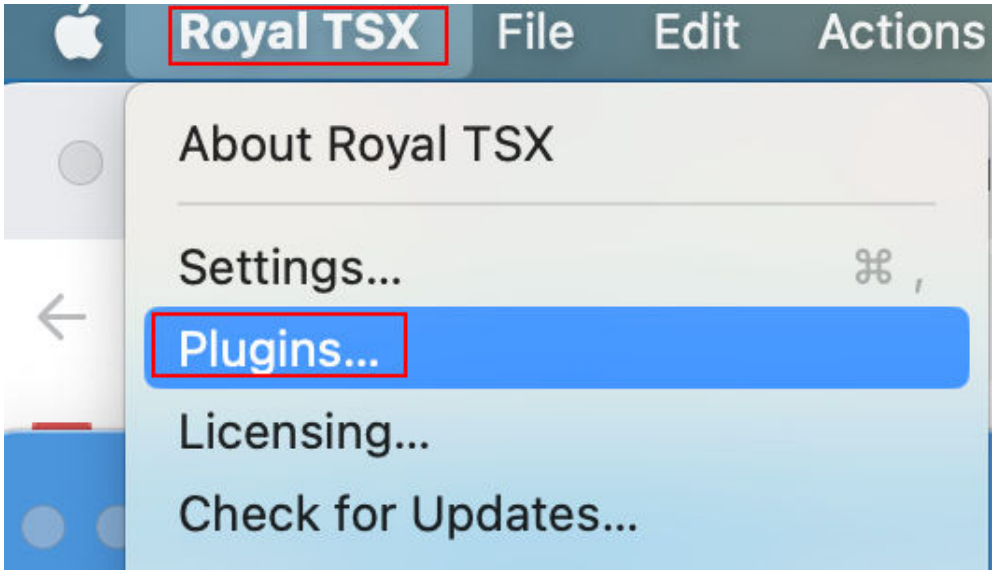
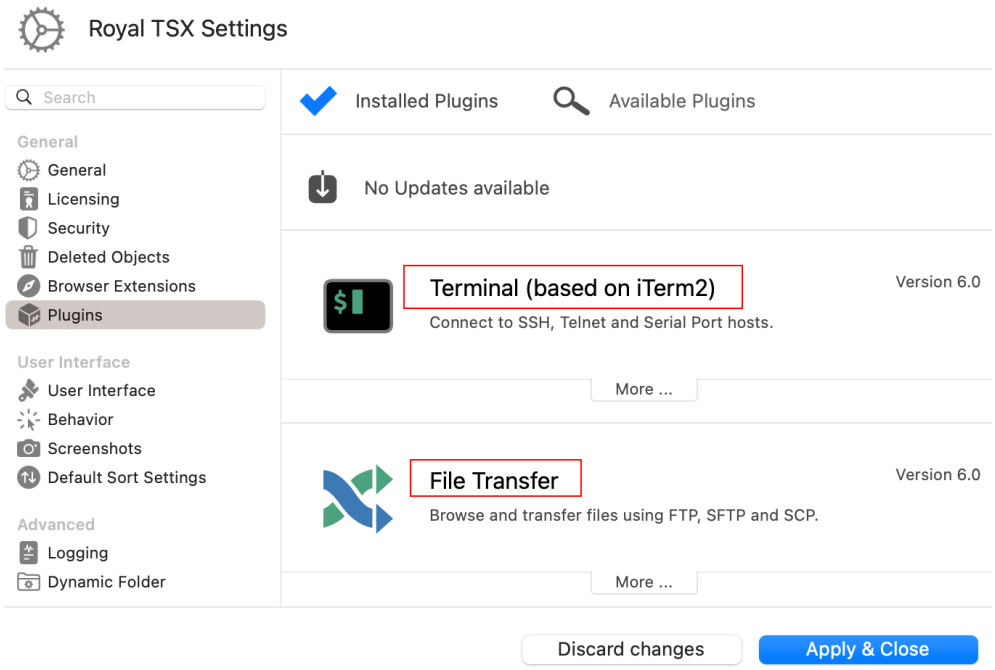
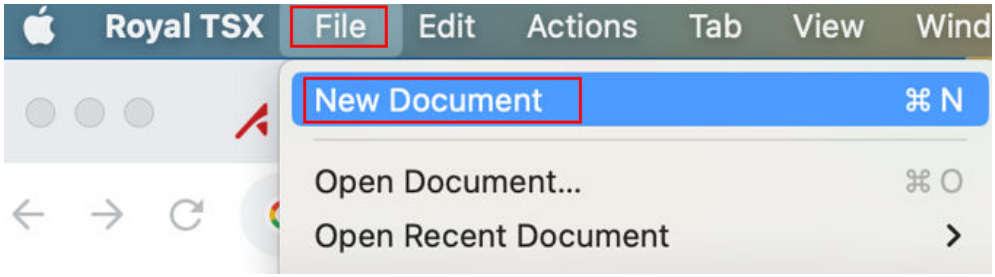


图 13-4 安装 Plugins



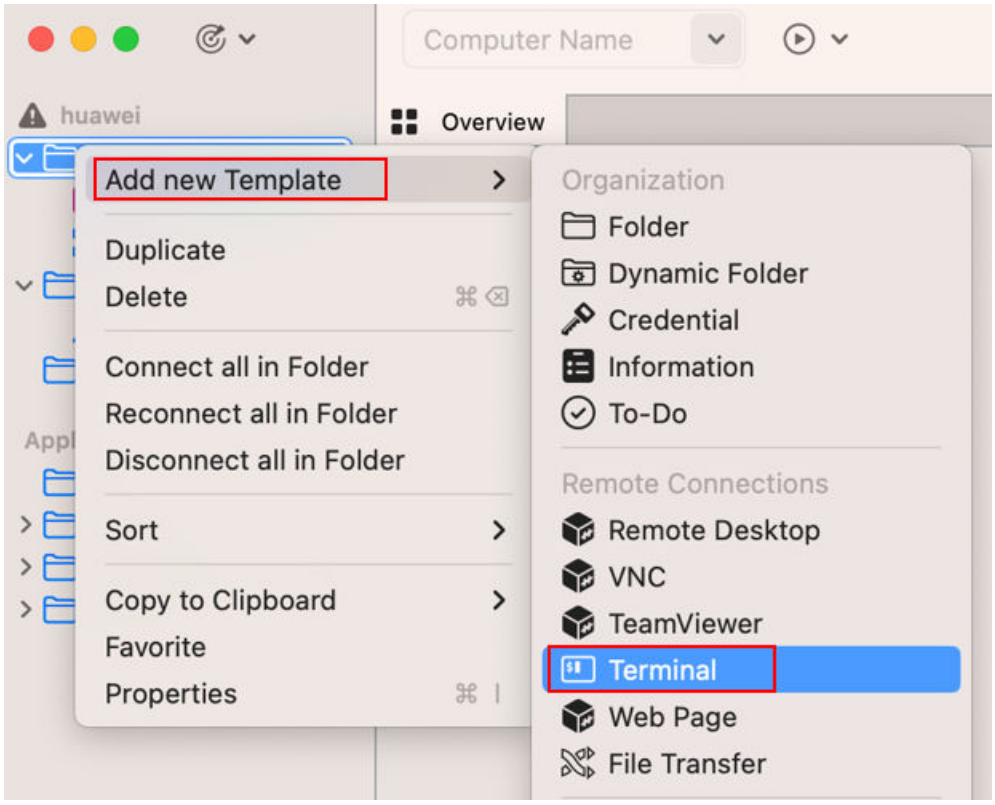
步骤5 单击界面上方菜单栏中的“File > New Document”，创建用于远程连接的Document，如图13-5所示。为方便识别Document类型，用户可自行修改Document名称。

图 13-5 创建 Document



步骤6 选中新建的Document，在弹框中选择“Add new Template > Terminal”，如图13-6所示，进入Terminal Connection Settings配置界面。

图 13-6 Add new Template



步骤7 在Terminal Connection Settings界面配置SSH连接信息。

1. Terminal页签：配置远端服务器连接信息，如图13-7所示，配置参数说明如表13-1所示。

图 13-7 Terminal 配置

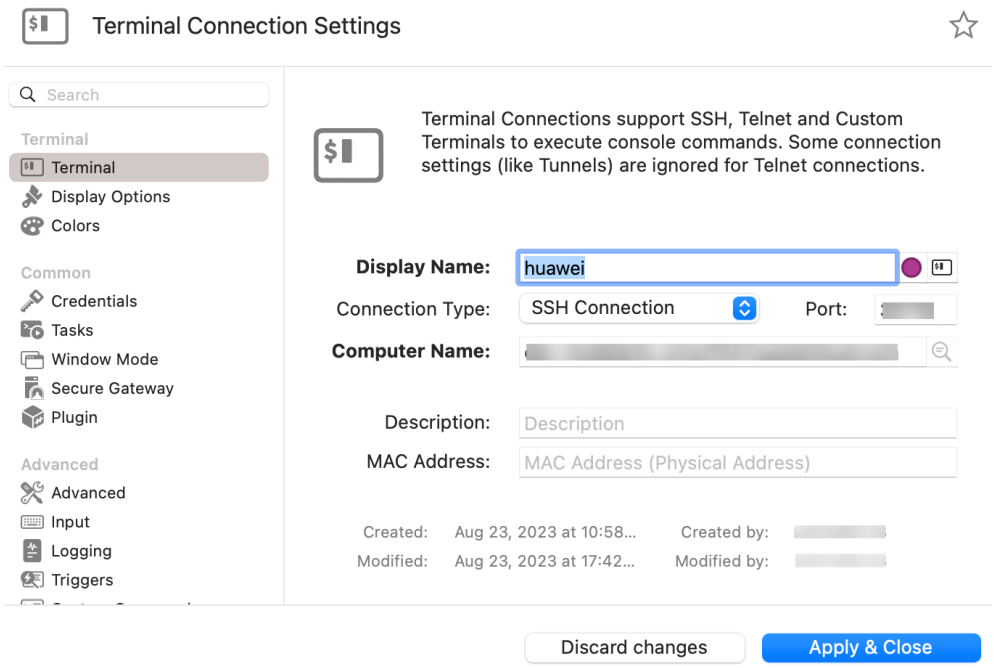


表 13-1 Terminal 配置参数说明

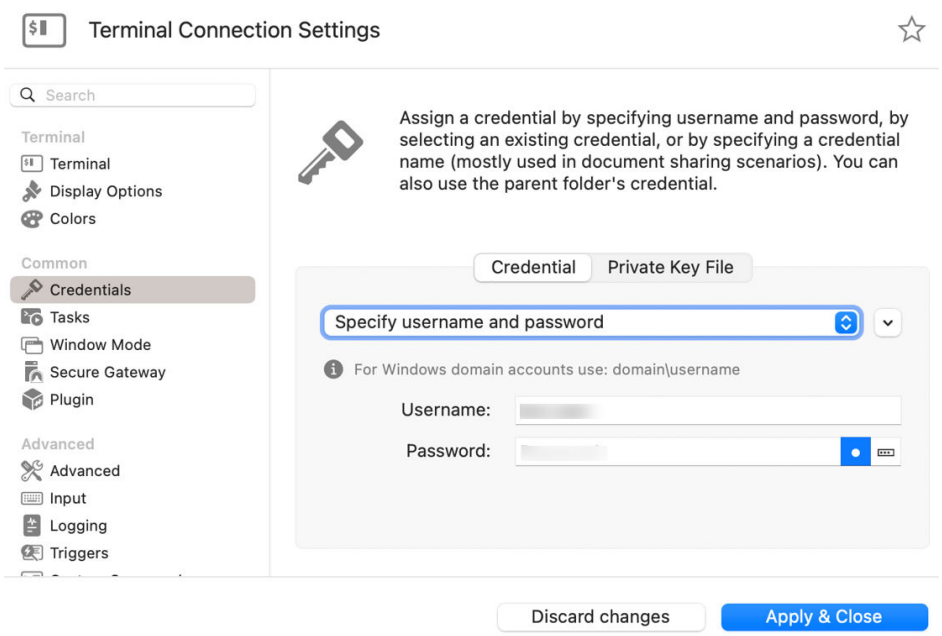
参数	说明	是否必填
Display Name	本地远程连接名称，自行填写。	是
Connection Type	连接方式，使用SSH Connection连接。	是
Port	远端服务器端口号。	是
Computer Name	远端服务器域名或者IP地址。	是
Description	关于该远程连接的描述信息，自行填写。	否
MAC Address	远端服务器的MAC地址。	否

配置完成后单击“Apply&Close”使其生效。

2. Credentials页签：配置登录方式，两种方式任选其一。
   - Credential：通过用户名和密码验证登录。

     在下方输入框选择“Specify username and password”，并在“Username”和“Password”栏中输入远端服务器的用户名和密码，如图13-8所示。配置完成后单击“Apply&Close”使其生效。

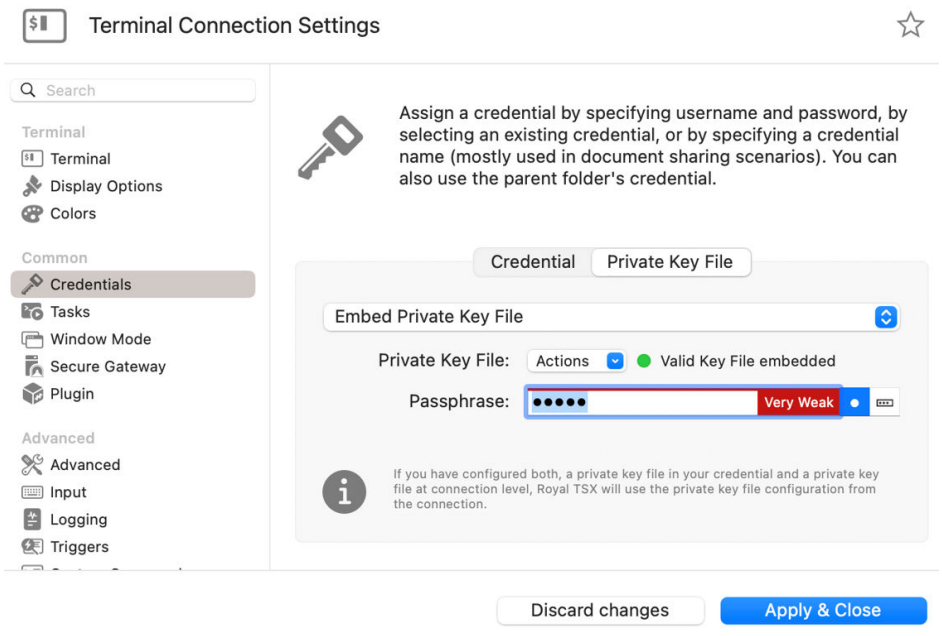
图 13-8 用户名和密码验证登录



– Private Key File：通过密钥验证登录。

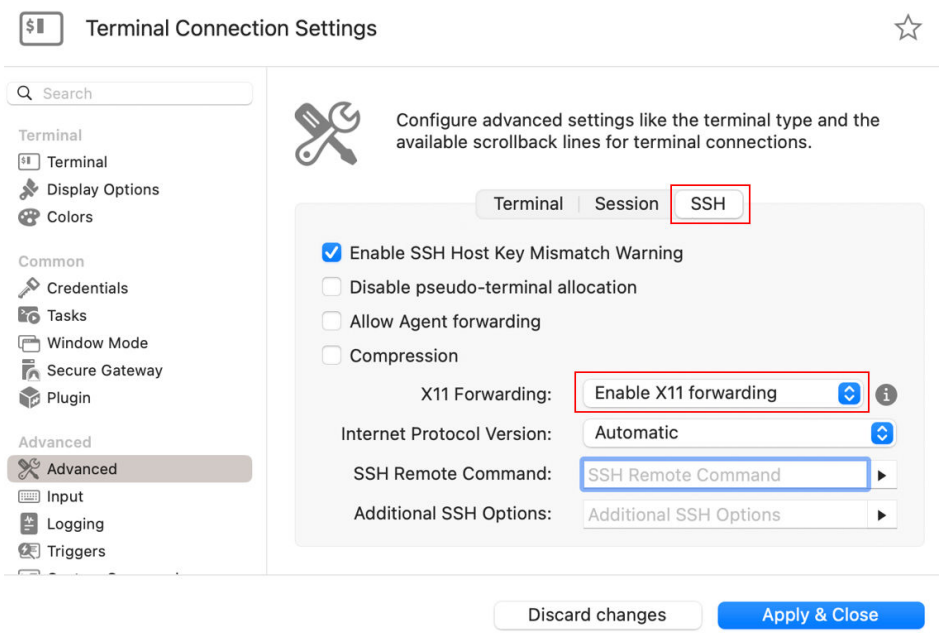
在下方输入框选择“Embed Private Key File”，并在“Private Key File”和“Passphrase”栏中选择密钥文件和输入密语或选择后方 随机生成密语，如图13-9所示。配置完成后单击“Apply&Close”使其生效。

图 13-9 密钥验证登录



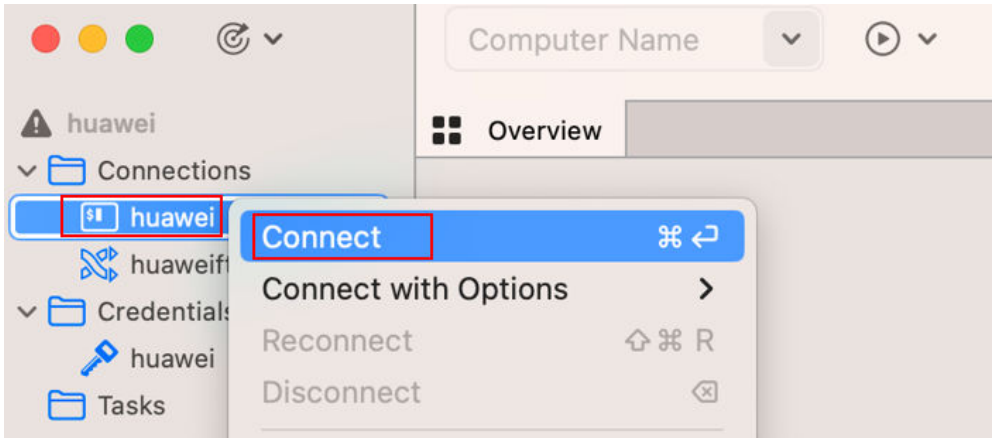
3. Advanced页签：单击“SSH”，在“X11 Forwarding”栏中选择“Enable X11 forwarding”，其他参数保持默认配置，如图13-10所示，配置完成后单击“Apply&Close”使其生效。

图 13-10 Advanced 配置



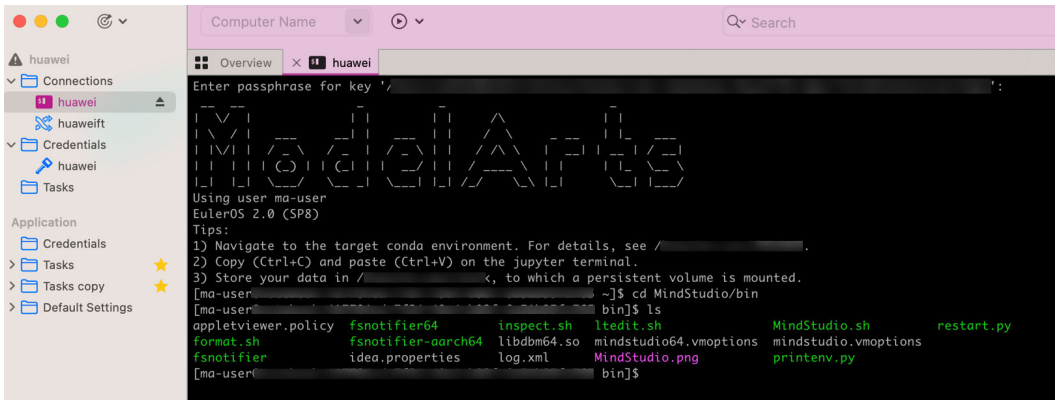
步骤8 SSH连接信息配置完成后，在步骤**步骤5**中新建的Document下会生成一个配置好的远程连接，选中该连接，在弹出框单击Connect，如**图13-11**所示。

图 13-11 建立远程连接



成功连接远程服务器，如**图13-12**所示。

图 13-12 连接成功



步骤9 进入MindStudio安装目录中的bin文件夹，执行如下命令启动MindStudio。

```
./MindStudio.sh
```

----结束

13.2 SSH 连接管理

概述

SSH连接是MindStudio与远程服务器建立远程登录会话的功能。SSH连接安全性高并且可以进行数据传输，便于MindStudio某些计算量较大的功能可以通过SSH连接来共享远端服务器的计算资源。

使用约束

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。目前支持算法：hmac-sha2-256、hmac-sha2-512、hmac-sha2-256-etm@openssh.com以及hmac-sha2-512-etm@openssh.com。
  - 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。禁止CBC模式的加密算法用于SSH2.0协议中（如AES128-CBC、AES256-CBC）。目前支持算法：aes128-gcm@openssh.com和aes256-gcm@openssh.com。
  - HostkeyAlgorithms：指定SSH-2允许使用的身份认证公钥算法，多个算法之间使用逗号分隔。目前支持算法：ssh-ed25519（要求OpenSSH6.5级以上版本）、rsa-sha2-512以及rsa-sha2-256。其中rsa的长度需大于等于3072bits。
  - KexAlgorithms：指定SSH-2允许使用的密钥加密算法，多个算法之间使用逗号分隔。目前支持算法：curve25519-sha256、curve25519-sha256@libssh.org以及diffie-hellman-group-exchange-sha256。OpenSSH6.7开始支持curve25519密钥交换算法。
  - PermitRootLogin：是否允许root登录。建议禁止root用户可以直接登录。

- MindStudio对工程内文件类型不做校验，需要确认上传文件的文件类型正确。
- 为了MindStudio客户端能连接，服务端需要配置如下算法：

  ciphers aes128-gcm@openssh.com,aes256-gcm@openssh.com

  macs hmac-sha2-256,hmac-sha2-512,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com

  hostkeyalgorithms ssh-ed25519,rsa-sha2-512,rsa-sha2-256

  kexalgorithms curve25519-sha256,curve25519-sha256@libssh.org,diffie-hellman-group-exchange-sha256

  服务端修改方式参考如下：

  a. 执行如下命令编辑配置文件。
  ```
  vi /etc/ssh/sshd_config
  ```
  请用户修改文件对应行，如果找不到对应行，则新增即可。

  b. 重启ssh服务。

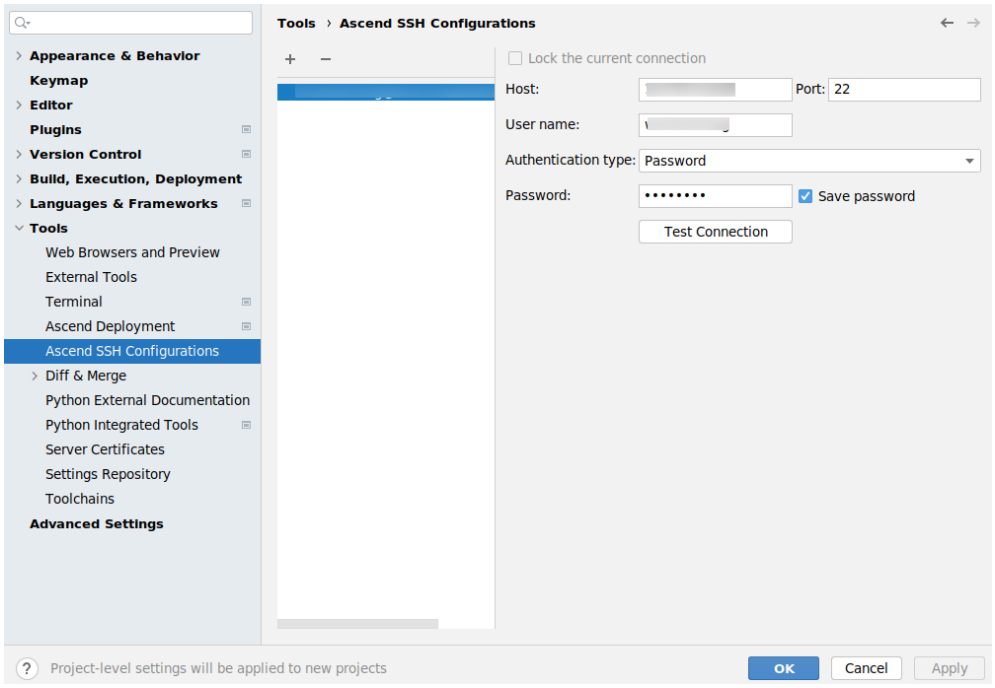
  - Red Hat和OpenEuler系列的操作系统
  ```
  service sshd restart
  ```
  - Ubuntu系列的操作系统
  ```
  service ssh restart
  ```
- 请保存好SSH用户名密码，不要共享给不信任的用户，以免造成环境破坏等影响。
- 为防止SSH暴力破解，建议用户在SSH端设置登录控制策略。

SSH 配置入口

两种进入SSH配置界面方式如下所示，SSH配置界面如**图13-13**所示，界面参数解释如**表13-2**所示。

- 在菜单栏依次选择“File > Settings... > Tools > Ascend SSH Configurations”进入SSH连接配置。
- 在欢迎界面依次选择“Customize > All Settings... > Tools > Ascend SSH Configurations”进入SSH连接配置。

图 13-13 SSH 配置



说明

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>，右键单击可修改别名。单击选中别名，可以在右侧详细面板中查看参数配置。

表 13-2 参数及图标说明

参数及图标	说明
Lock the current connection	选项置灰，用户不可操作。如果SSH连接被添加到集群管理中，则该选项会自动勾选，SSH连接被锁定，不可修改和删除。
Host	连接目标地址（IP或域名）。 IP支持IPv4和IPv6。 **说明** 不支持以fe80开头的IPv6地址，fe80开头的IPv6地址是link-local地址，只能在子网使用，且通过该地址登录设备需要输入IP对应的网口名，不符合MindStudio中SSH连接界面的输入格式。
Port	连接目标地址的端口号，默认值：22。
User name	登录目标地址的用户名。
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，SSH密钥申请方式请参见**申请密钥对**。

参数及图标	说明
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。 **说明** 如果没勾选Save password。那么密码自动会在24小时后清除，届时再次使用到SSH会弹出密码输入框，需要再次输入密码方能执行SSH任务。
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 password复选框可以保存密语。
Test Connection	单击该按钮测试连接是否成功。
+	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。

申请密钥对

密钥对可以在本地服务器申请也可以在目标服务器申请，这里以本地服务器申请为例进行描述。

步骤1 登录本地服务器，使用如下命令申请密钥对。

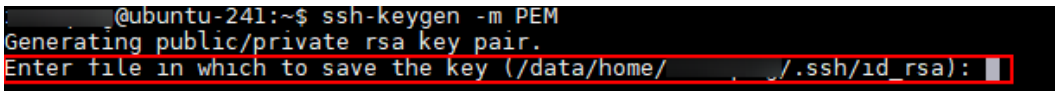
```
ssh-keygen -m PEM
```

或者

```
ssh-keygen -t rsa -m PEM
```

步骤2 设置密钥对存放路径，如图13-14所示。

图 13-14 配置密钥对存放路径

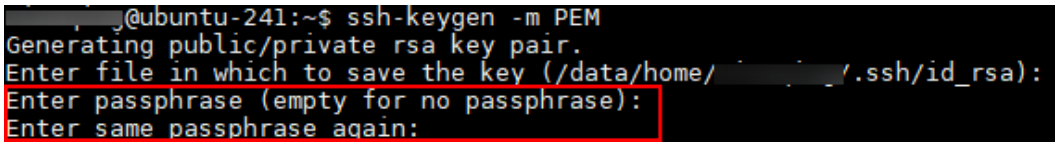


说明

可以不设置存放路径，直接按回车键，则密钥对会存放在默认路径（*$home*/.ssh/）。

步骤3 设置及确认密语，如图13-15所示。

图 13-15 设置密语



> **说明**
>
> 建议passphrase不为空且密语长度至少为5个字符。

设置完成后，按回车键，在默认路径（*$home*/.ssh/）中生成私钥文件（id_rsa）和公钥文件（id_rsa.pub）。

步骤4 使用以下命令将公钥复制至目标服务器，用于免用户密码校验登录。

```
ssh-copy-id -i ~/.ssh/id_rsa.pub -p 22 目标服务器用户名@目标服务器IP地址
```

> **说明**
>
> 公钥复制在目标服务器的*$home*/.ssh/authorized_keys文件中。

----结束

新增 SSH 连接

步骤1 参见**SSH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**表13-2**配置参数后，单击“OK”或“Apply”。

----结束

删除 SSH 连接

步骤1 参见**SSH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

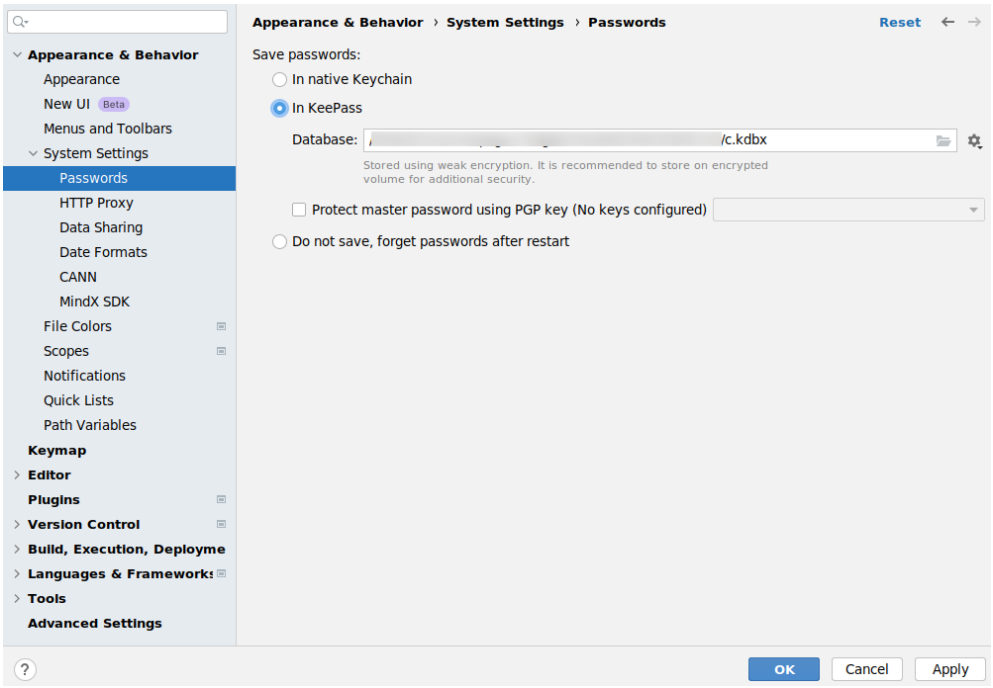
保存 SSH 密码

步骤1 参见**SSH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 13-16 密码保存方式界面



步骤3 选择密码保存方式。

- In KeePass
- Do not save，forget passwords after restart

说明

两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。

步骤4 单击“OK”或“Apply”。

----结束

开启 SSH Remote Terminal 窗口

通过开启SSH Remote Terminal窗口功能，可以在MindStudio IDE界面上对SSH连接环境进行命令行操作。

说明

已有配置好的SSH连接，详细操作请参见**新增SSH连接**。

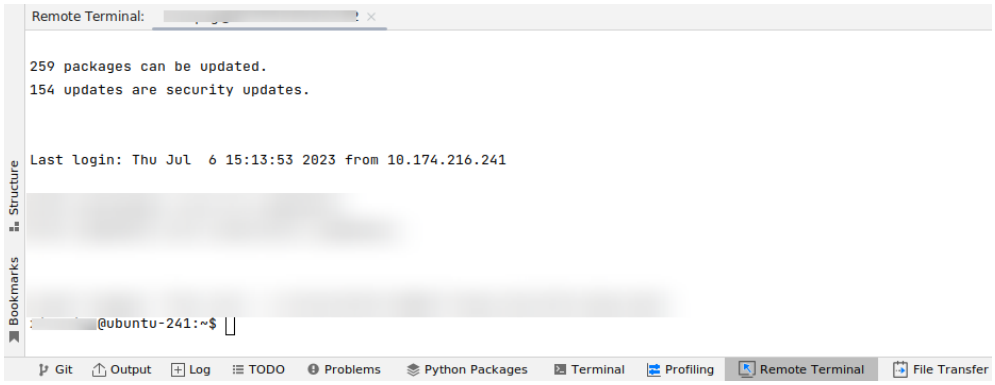
步骤1 在菜单栏选择“Tools > Start Ascend SSH session...”。

弹出“Select Host to Connect”弹窗。

步骤2 单击需要开启Remote Terminal窗口的SSH连接。

在MindStudio IDE界面下方显示远端窗口信息，可直接进行命令行操作。

图 13-17 Remote Terminal



说明

为防止SSH长时间保持会话可能存在的安全风险，建议用户在服务端设置SSH会话超时时间。

----结束

13.3 Ascend Deployment

通过Ascend Deployment功能可以将指定的文件、文件夹同步到远程指定机器的指定目录。

配置 Ascend Deployment 功能

步骤1 在菜单栏选择“File > Settings…”。

在左边的导航栏选择“Tools > Ascend Deployment”，进入图13-18。

图 13-18 Ascend Deployment 界面

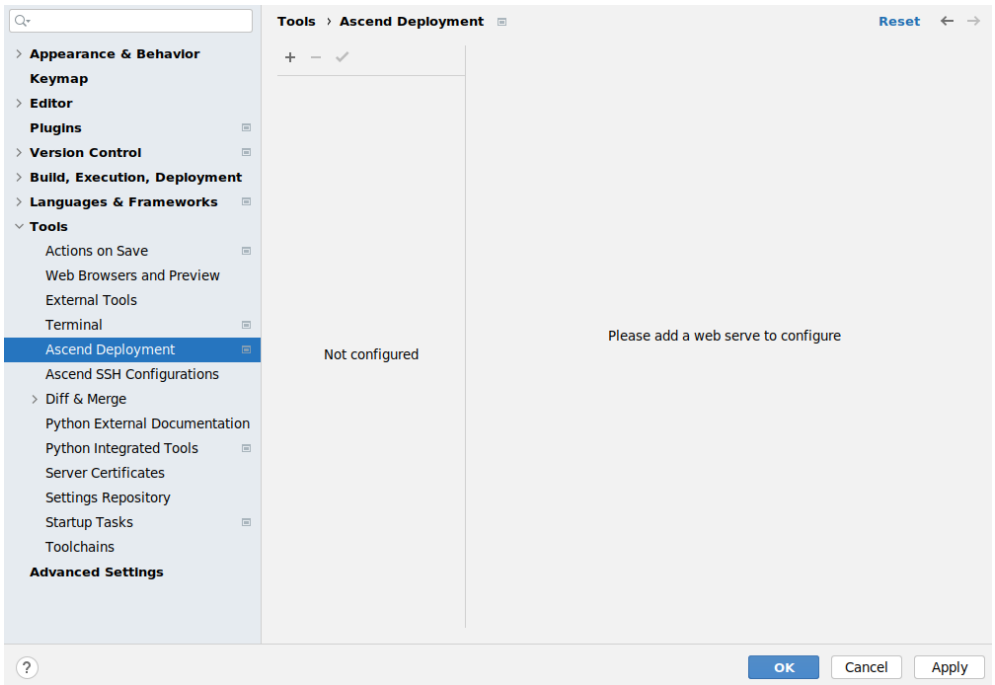


表 13-3 参数及图标说明

参数及图标	说明
+	新增Deployment。
−	删除Deployment，选中需要删除的Deployment，单击该按钮删除。
✓	指定选中的Deployment为默认Deployment，单击该按钮。再单击一次该按钮，即可取消默认。

步骤2 配置远程服务器连接。

单击+，将弹出新建服务器命名窗口，输入服务器名字，单击“OK”，界面将显示需要配置的“Connetion”、“Mappings”及“Excluded Paths”。

图 13-19 Connection

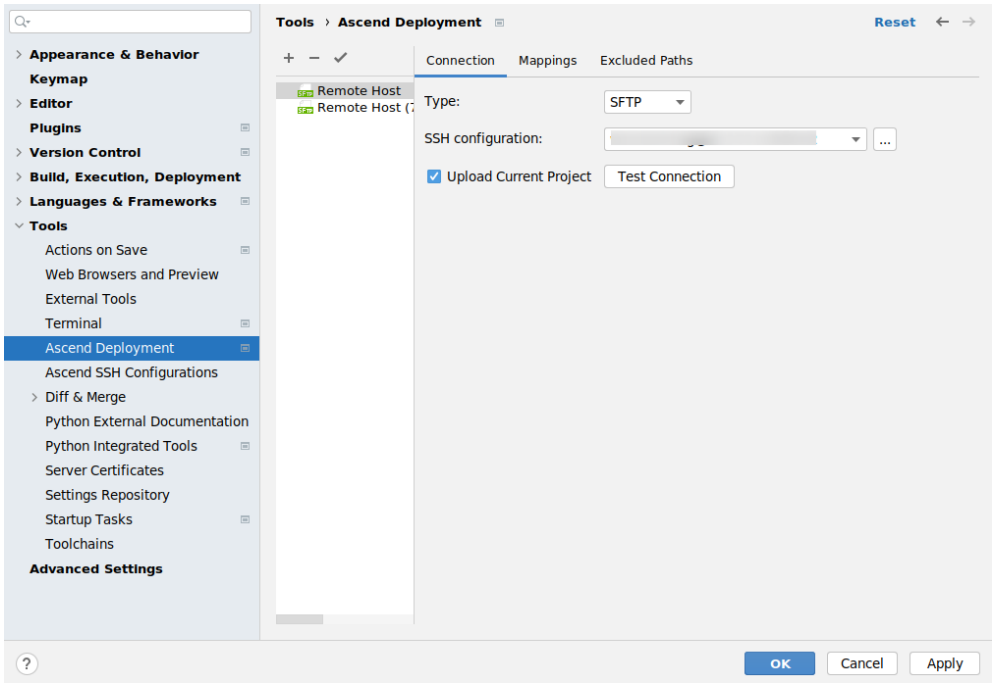


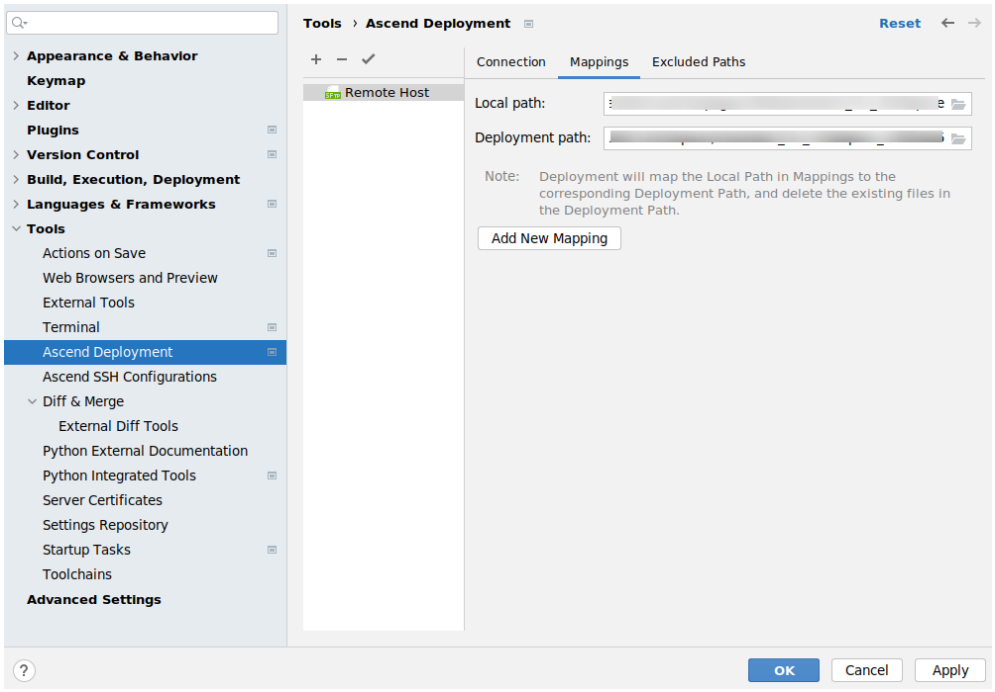
表 13-4 参数及图标说明

参数及图标	说明
Type	文件传输协议类型。目前仅支持SFTP。
SSH configuration	SSH远程连接。单击 ▼ 或 ··· 可选择已配置的SSH连接，如果没有需要的SSH连接，可单击 ···，在弹出的窗口中的+图标配置SSH连接。

参数及图标	说明
Upload Current Project	同步当前工程至远程服务器。 选中此按钮，单击Apply，将会自动同步当前工程至远端服务器，并且默认开启**Automatic Upload**（自动上传功能）。
Test Connection	单击该按钮测试连接是否成功。

步骤3 （必选）配置映射路径关系。

图 13-20 Mappings



如需继续配置多条Mapping，可单击“Add New Mapping”进入多条配置界面，继续添加。

图 13-21 配置多条 Mapping

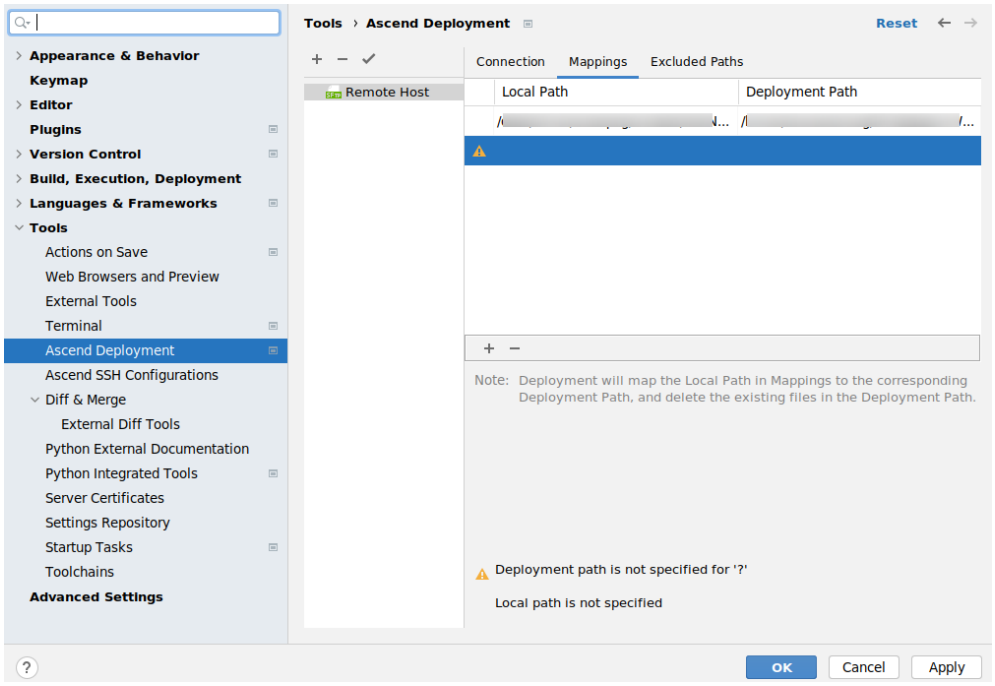


表 13-5 参数及图标说明

参数及图标	说明
+	新增映射路径关系信息。
−	删除映射路径信息。选中需要删除的路径映射关系信息，再单击此按钮，即可删除指定的路径映射关系信息。
Local Path	本地项目文件夹路径。单击对应文本框，将自动识别已打开的项目文件夹路径，也可通过在文本框中手动输入或单击文本框右侧文件夹图标进行配置。 取值示例如下： • Windows：C:\Users\username\MindstudioProjects\MyApp • Linux：/home/username/MindstudioProjects/MyApp
Deployment Path	本地项目文件夹路径映射到远端服务器中的路径。可通过在文本框中手动输入或单击文本框右侧文件夹图标进行配置。

步骤4 （可选）配置排除路径信息，用户根据需要配置。

图 13-22 Excluded Paths

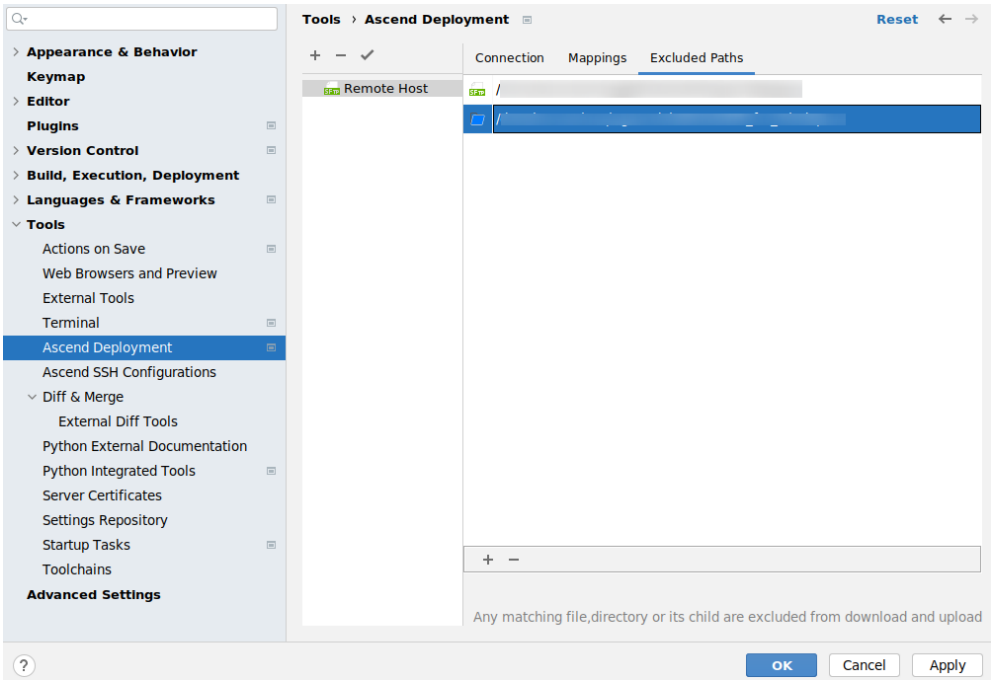


表 13-6 参数及图标说明

参数及图标	说明
+	新增排除路径信息。单击此按钮，可选择“Deployment Path”或“Local Path”，分别对应SFTP和图标。
−	删除排除路径信息。选中需要删除的排除路径信息，再单击此按钮，即可删除指定的路径映射关系信息。
SFTP	排除远端服务器中的路径。通过在文本框手动输入或单击文件夹图标进行配置，配置生效后，将排除对该路径下的所有子目录文件和本目录文件进行上传、下载操作。
	排除本地路径。通过在文本框手动输入或单击文件夹图标进行配置，配置生效后，将排除该路径下的所有子目录文件和本目录文件进行上传、下载操作。 取值示例如下： • Windows：C:\Users\username\MindstudioProjects\MyApp • Linux：/home/username/MindstudioProjects/MyApp

步骤5 单击“Apply”，Deployment配置生效，在使用run/debug等功能时，会触发Deployment同步操作。

----结束

使用 Ascend Deployment 功能

如果需要使用Ascend Deployment功能，可通过如下操作实现。

- 自动上传
  a. 在菜单栏选择“File > Settings…”。
     在左边的导航栏选择“Tools > Ascend Deployment”进入Ascend Deployment界面，在Deployment列表选中需要使用的Deployment，然后单击上方的✔将其设置为默认Deployment。
  b. 在菜单栏选择“Tools > Ascend Deployment > Automatic Upload”。
- 手动同步
  - 在左侧工程目录中选中需同步的文件或目录，在菜单栏选择“Tools > Ascend Deployment”，单击“Upload to...”或者“Download from...”手动触发Deployment的上传或者下载功能。
  - 在左侧工程目录中选中需同步的文件或目录，单击鼠标右键选择“Ascend Deployment”，单击“Upload to...”或者“Download from...”手动触发Deployment的上传或者下载功能。

⚠ 注意

- 针对上传过程，Deployment会将mapping中的“Local Path”完全映射于对应的“Deployment Path”，若本地有a、b和c文件，而远端“Deployment Path”中有d文件，上传的结果是远端对应目录下只有a、b和c文件，d文件会被删除。
- 针对下载过程，不会使用完全映射过程，如远端有a、b、c文件，本地有d文件，下载的结果是本地对应目录存在a、b、c和d文件。

13.4 Toolchains

通过Toolchains功能可以快速配置C/C++工程本地和远程编译的功能，支持gcc和LLVM编译工具。

须知

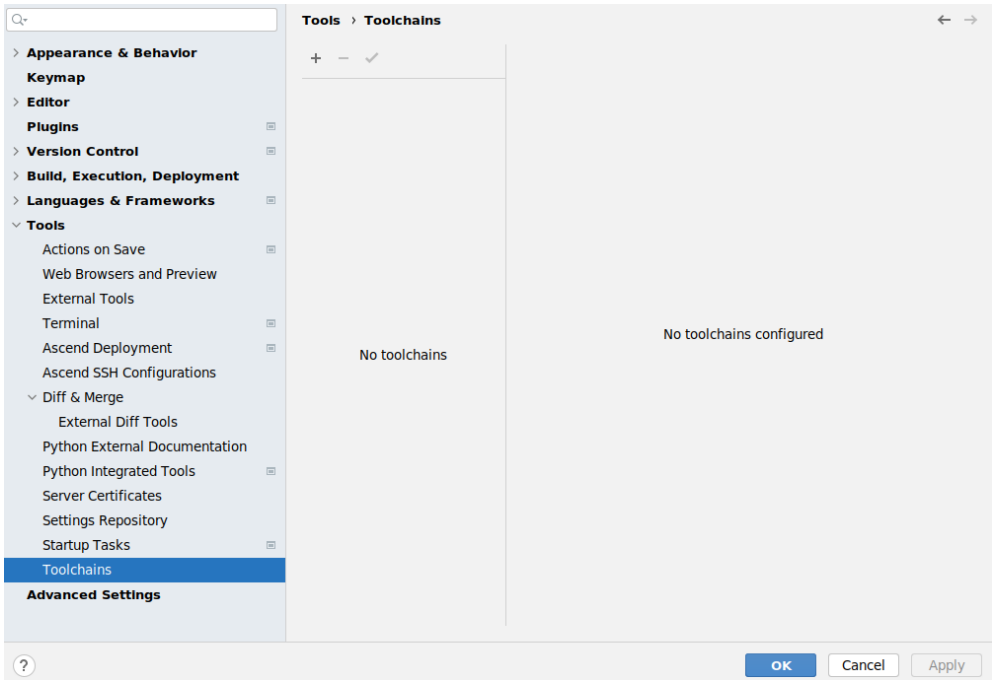
工程在运行时，将根据编译过程时使用的Toolchain配置进行远端或本地运行。（如：编译时使用本地Toolchain配置，则该工程运行时，将默认在本地执行。）

本地编译配置

步骤1 在菜单栏选择“File > Settings...”。

在左边的导航栏选择“Tools > Toolchains”，进入图13-23。

图 13-23 配置工具链界面



步骤2 单击“+”按钮，添加Toolchains配置项。

- Linux配置工具链为“System”，如图13-24所示。
- Windows配置工具链为“MinGW”，如图13-25所示。
- LLVM工具链配置，Linux请参考图13-26进行配置；Windows请参考图13-27进行配置。

图 13-24 Linux 配置工具链 System

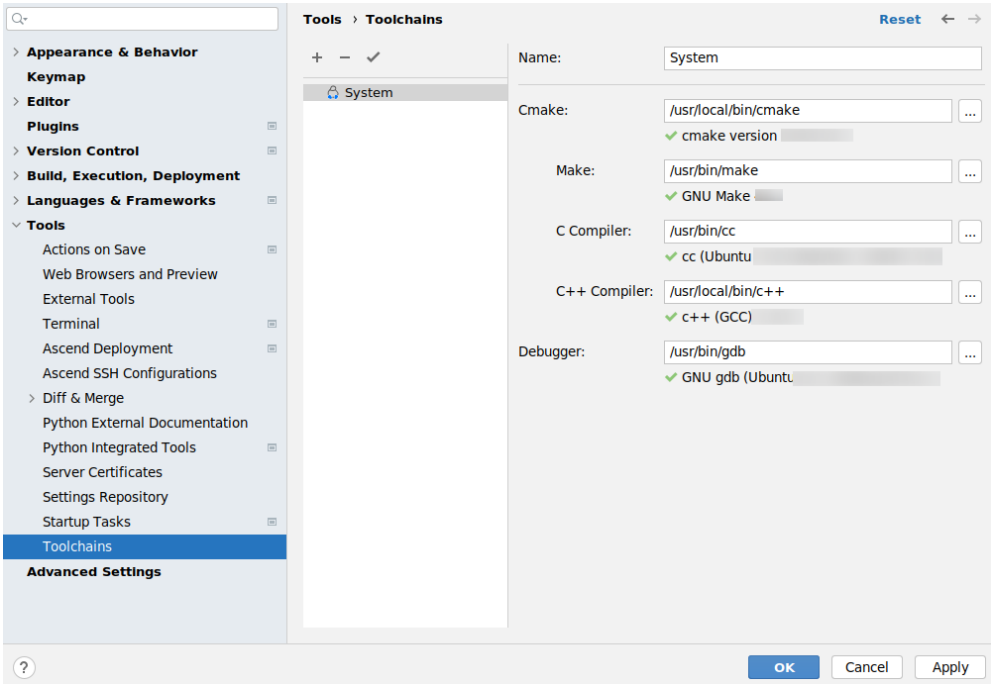


图 13-25 Windows 配置工具链 MinGW

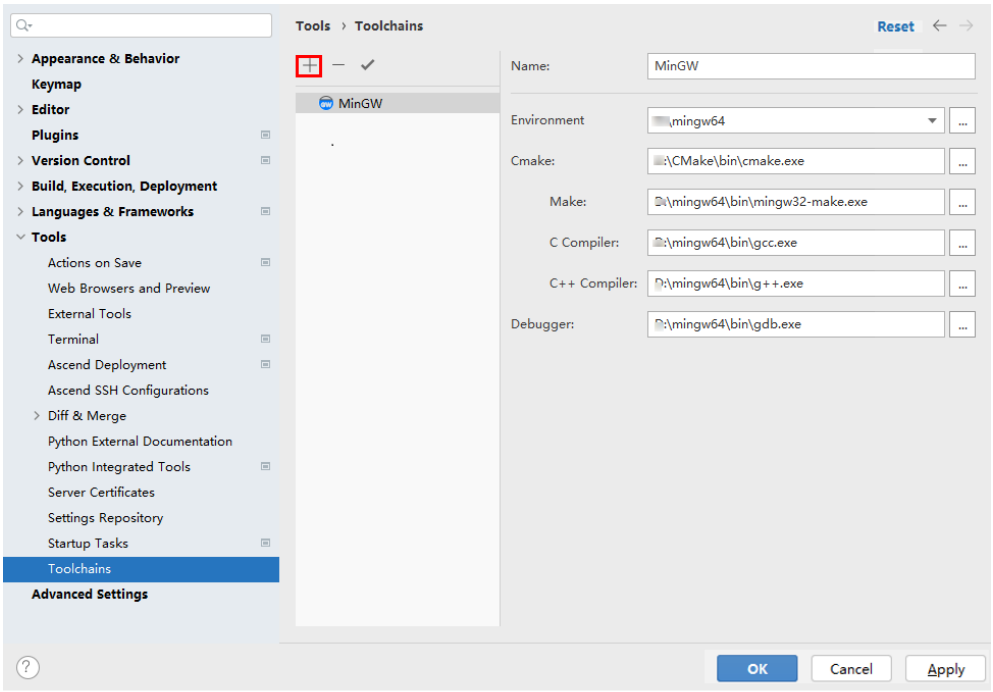


图 13-26 Linux 配置工具链 LLVM

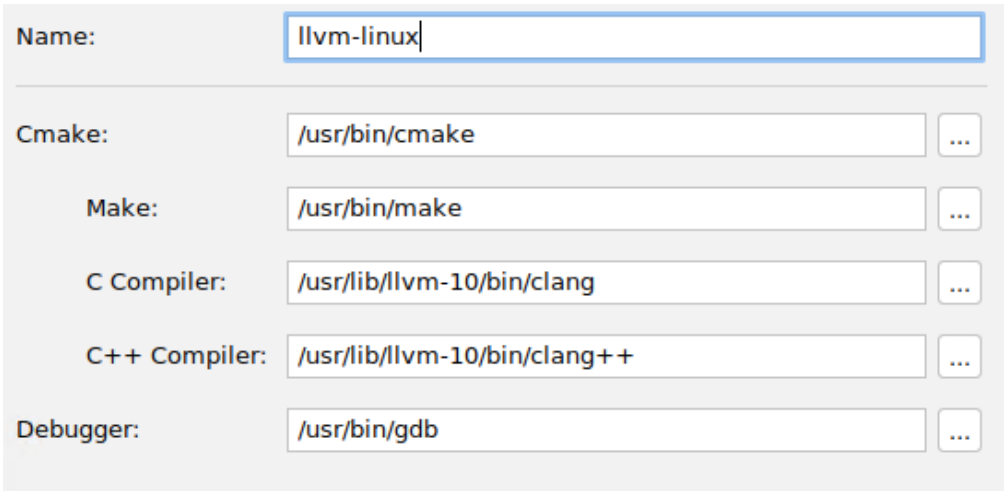


图 13-27 Windows 配置工具链 LLVM

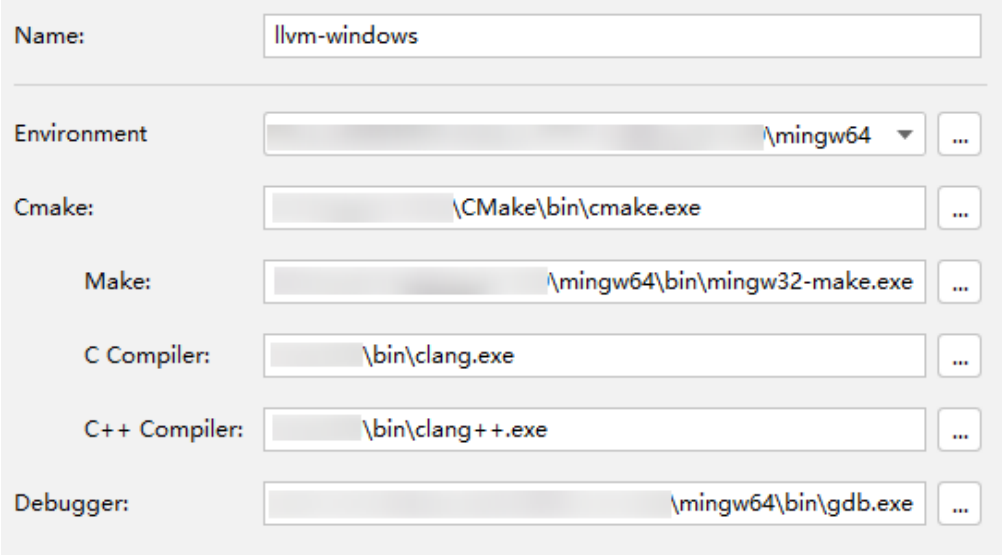


表 13-7 参数及图标说明

参数及图标	说明
Name	Toolchains配置名称。
Environment	mingw的路径，配置环境变量后可自动识别，也可通过手动输入或单击右侧…进行配置。 **说明** Linux环境下无此选项。
Cmake	本地服务器中的Cmake路径，也可通过手动输入或单击右侧…进行配置。
Make	本地服务器中的Make路径，也可通过手动输入或单击右侧…进行配置。
C Compiler	本地服务器中的C Compiler路径，也可通过手动输入或单击右侧…进行配置。
C++ Compiler	本地服务器中的C++ Compiler路径，也可通过手动输入或单击右侧…进行配置。
Debugger	本地服务器中的Debugger路径，也可通过手动输入或单击右侧…进行配置。
+	新增工具链配置。
−	删除工具链配置，选中需要删除的工具链配置，单击该按钮删除。

工具链生效后会在左边的导航栏“Tools > Toolchains”中自动生成一条信息，以供编译配置选择。

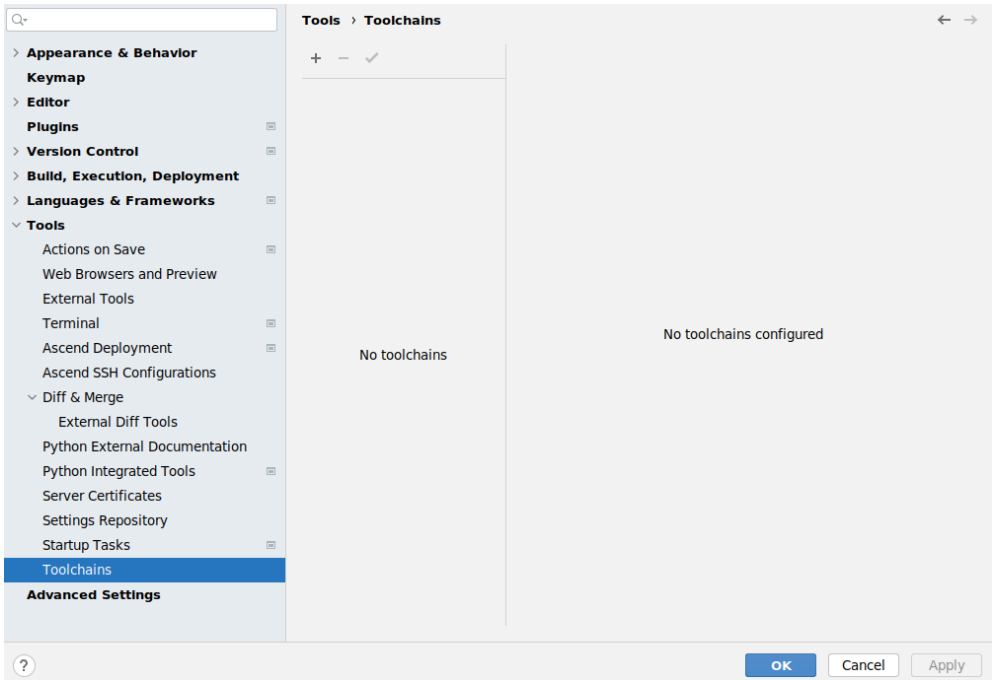
----结束

远程编译配置

步骤1 在菜单栏选择“File > Settings...”。

在左边的导航栏选择“Tools > Toolchains”，进入图13-28。

图 13-28 配置工具链界面



步骤2 单击“+”按钮，配置工具链“Remote Host”。

图 13-29 配置工具链 Remote Host

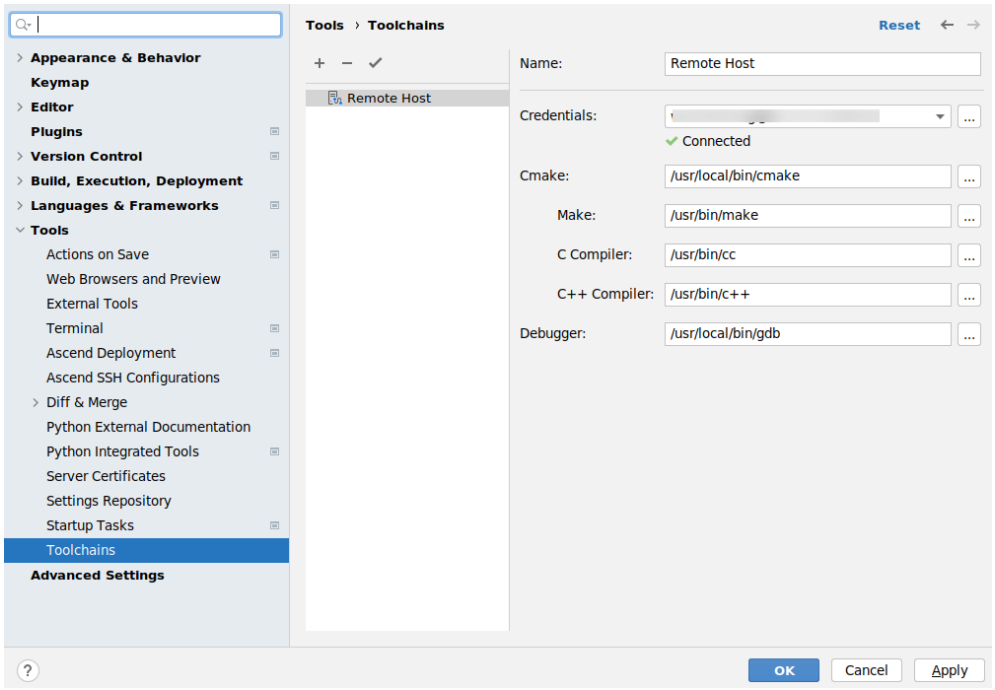


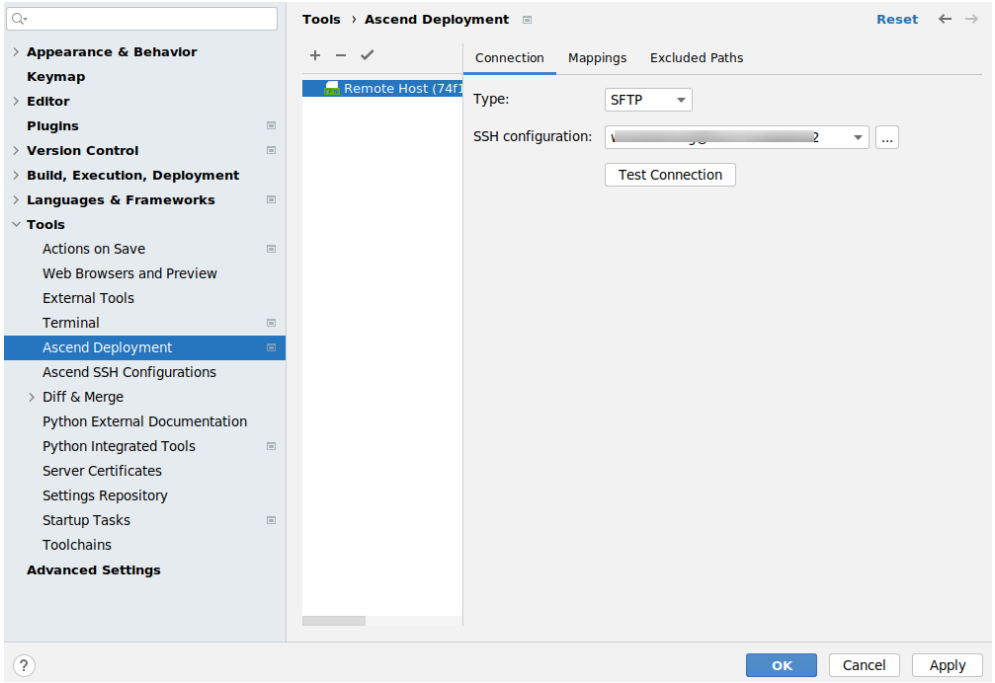
表 13-8 参数及图标说明

参数及图标	说明
Name	Toolchains配置名称。
Credentials	SSH配置，单击 ▼ 或 ··· 图标选择需要的SSH连接，如果没有需要的SSH连接，可单击弹出窗口中的 + 图标进行SSH连接配置。
Cmake	指定远端服务器中的Cmake路径，配置“Credentials”后自动识别，也可通过手动输入或单击右侧 ··· 进行配置。
Make	指定远端服务器中的Make路径，配置“Credentials”后自动识别，也可通过手动输入或单击右侧 ··· 进行配置。
C Compiler	指定远端服务器中的C Compiler路径，配置“Credentials”后自动识别，也可通过手动输入或单击右侧 ··· 进行配置。
C++ Compiler	指定远端服务器中的C++ Compiler路径，配置“Credentials”后自动识别，也可通过手动输入或单击右侧 ··· 进行配置。

参数及图标	说明
Debugger	指定远端服务器中的Debugger路径，配置“Credentials”后自动识别，也可通过手动输入或单击右侧 ... 进行配置。
+	新增工具链配置。
−	删除工具链配置，选中需要删除的工具链配置，单击该按钮删除。

工具链生效后会在左边的导航栏“Tools > Ascend Deployment”中自动生成一条Deployment信息，如图13-30所示。

图 13-30 生成 Deployment 信息



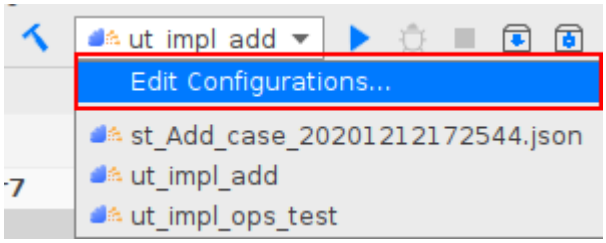
----结束

13.5 运行管理

13.5.1 修改运行任务配置

步骤1 在MindStudio主界面，在工具栏选择“Edit Configurations...”，如图13-31所示。

图 13-31 配置入口



步骤2 在界面左侧导航栏选择待修改的运行任务配置信息，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任务，如图13-32所示。

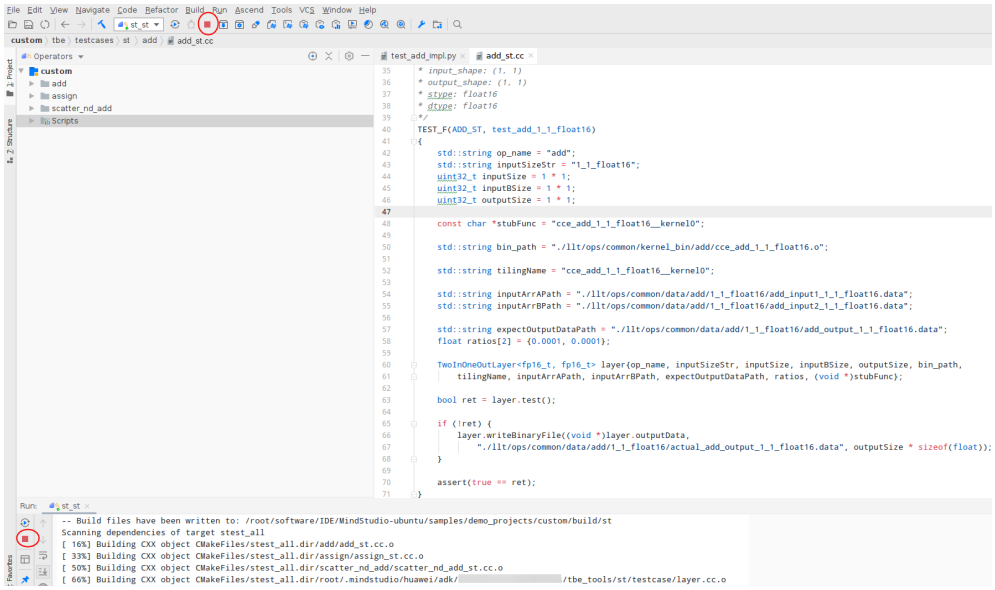
图 13-32 运行任务

----结束

13.5.2 停止运行/重新运行任务

如果想停止运行任务，可以单击图13-33所示的 按钮，从而查看运行日志打印。

图 13-33 运行停止按钮（任选其一操作）



如果想重新运行，可以单击 按钮。

13.6 CANN 管理

13.6.1 概述

CANN 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。

CANN管理入口：

- 在MindStudio快捷工具栏，单击如下图标进入CANN Manager界面。

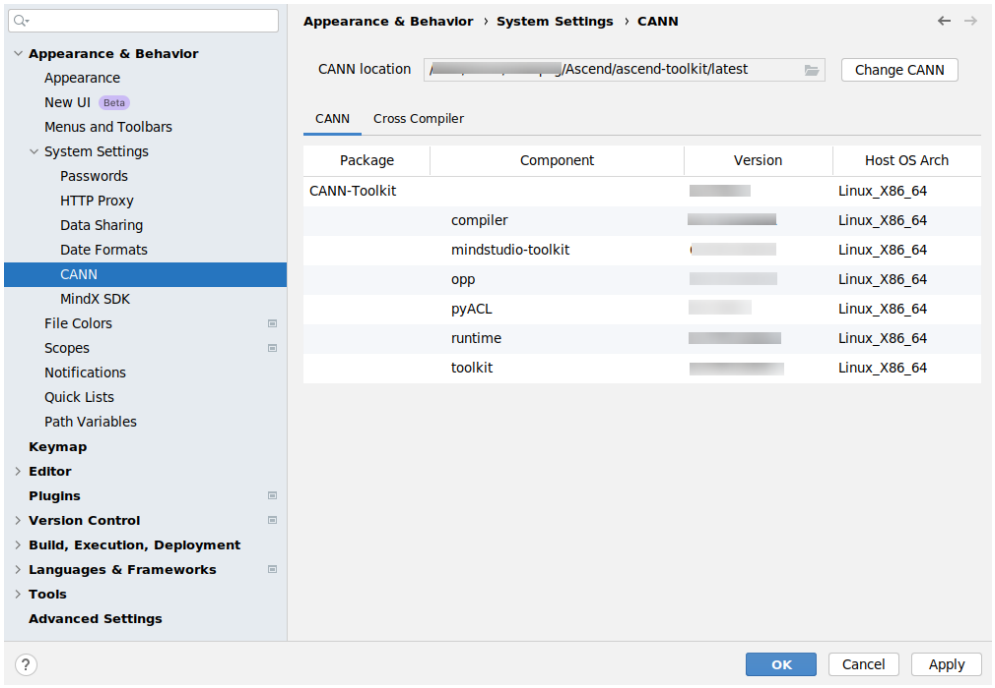
  图 13-34 快捷工具栏进入 CANN Manager

- 在MindStudio工程界面菜单栏依次选择“File > Settings... > Appearance&Behavior > System Settings > CANN”，弹出CANN Manager界面。
- 在MindStudio工程界面菜单栏依次选择“Ascend > CANN Manager”，弹出CANN Manager界面。
- 在MindStudio欢迎页面依次选择“Customize > All settings... > Appearance&Behavior > System Settings > CANN”，弹出CANN Manager界面。

以上4种方式打开的CANN Manager界面如图13-35或图13-36所示。

图 13-35 CANN Manager 界面（Linux 系统）



说明

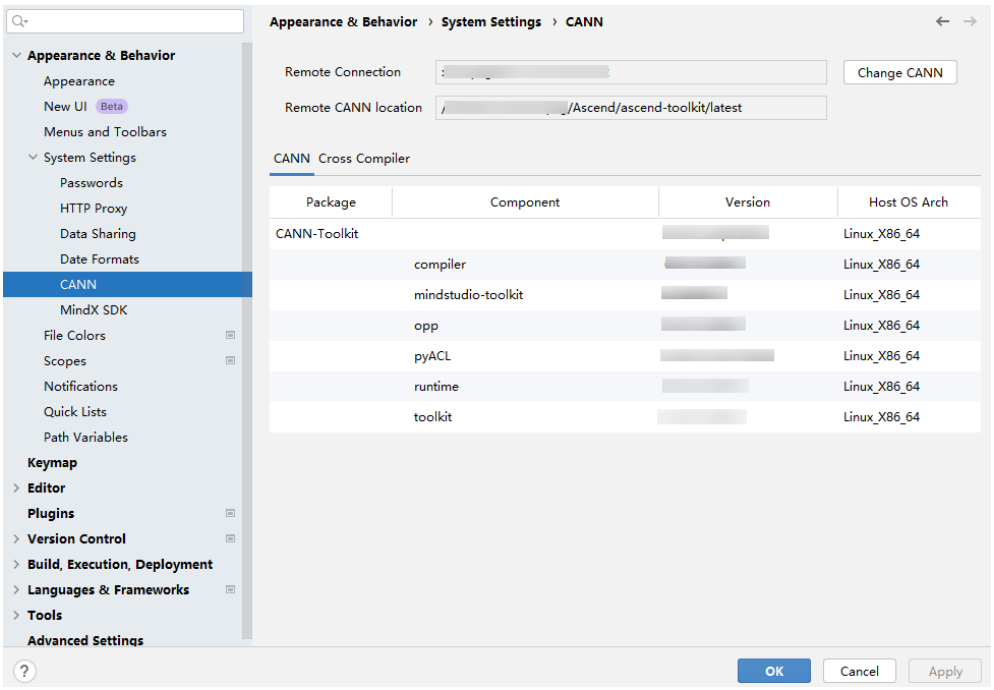
用户安装不同的CANN版本可能导致图13-35中的Component和Host OS Arch列显示与示例不同，具体显示请以实际界面为准。

界面参数以及图标解释如表13-9所示。

表 13-9 CANN Manager 界面参数以及图标说明（Linux 系统）

参数以及图标	说明
CANN location	Ascend-cann-toolkit开发套件包的安装路径，默认为$HOME/Ascend/ascend-toolkit/*{version}*，通过单击右侧的可以进行多版本CANN切换。
Change CANN	可以进行多版本Ascend-cann-toolkit开发套件包切换。
Package	CANN包名称。
Component	组件名称。
Version	CANN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。

图 13-36 CANN Manager 界面（Windows 系统）



界面参数以及图标解释如表13-10所示。

表 13-10 CANN Manager 界面参数以及图标说明（Windows 系统）

参数以及图标	说明
Remote Connection	Ascend-cann-toolkit开发套件包所在远程环境IP地址。
Remote CANN location	Ascend-cann-toolkit开发套件包的安装路径，默认为$HOME/Ascend/ascend-toolkit/*{sotftware version}*。

参数以及图标	说明
Change CANN	可以进行多版本Ascend-cann-toolkit开发套件包切换。
Package	CANN包名称。
Component	组件名称。
Version	CANN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。

13.6.2 切换/激活 CANN 包

切换 CANN

- **Linux系统：**
  a. 请参考“13.6.1 概述”进入如图CANN管理页面，通过以下两种方式进入如图13-37所示界面。
     - 单击CANN location右侧的，系统弹出Toolkit所在路径的页面。如图13-37所示。
     - 单击CANN管理页面中的“Change CANN”按钮，然后单击系统弹出的图13-38所示界面的。

图 13-37 选择 Toolkit 版本

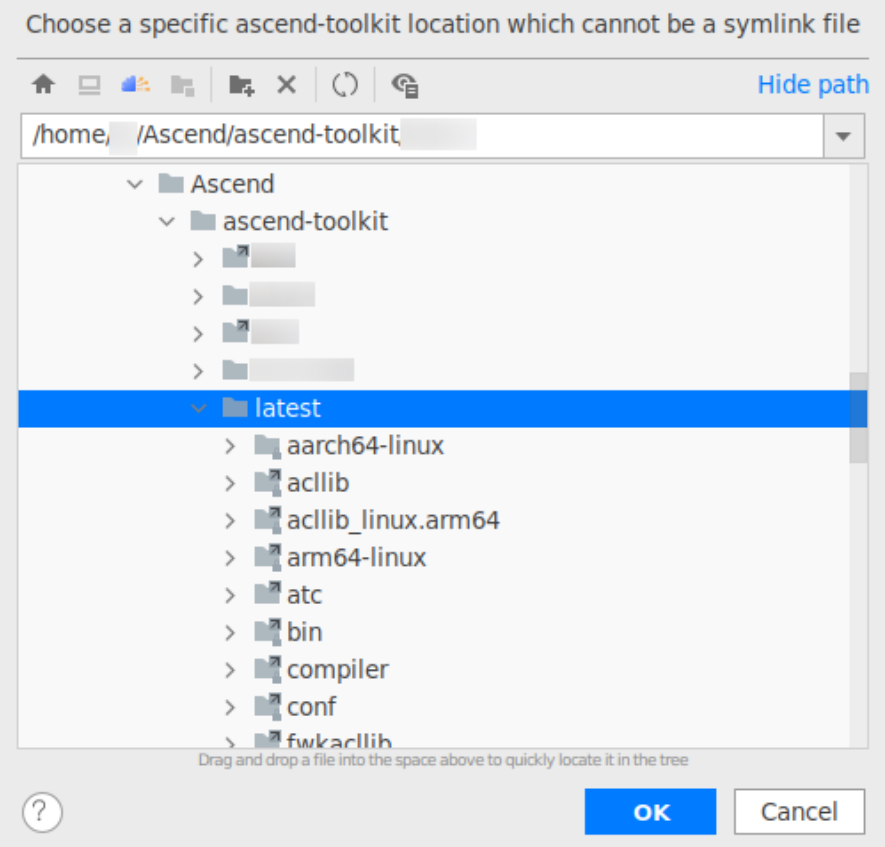
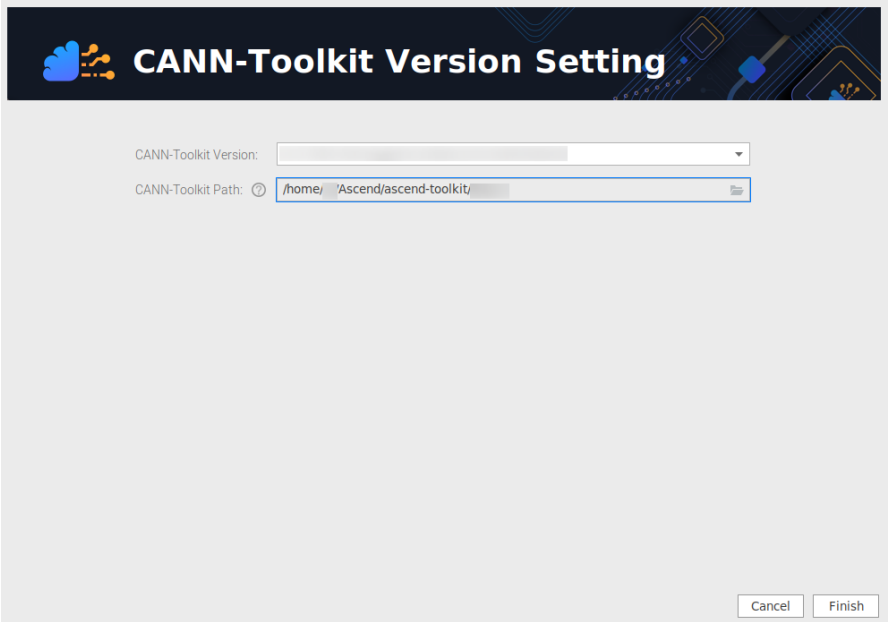
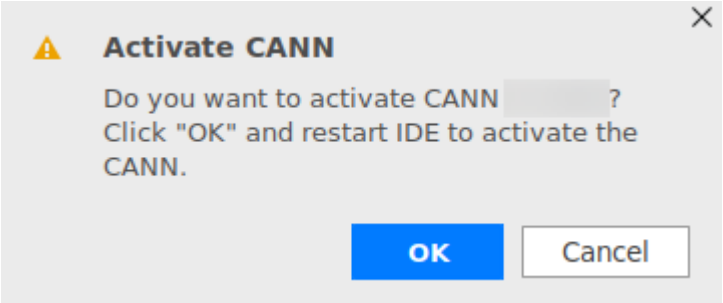


图 13-38 选择 Ascend-cann-toolkit 开发套件包安装路径



b. 选中需要替换的Toolkit版本，单击“OK”。弹出确认页面，如图13-39所示。

图 13-39 确认页面



i. 选择“OK”，CANN切换到新版本，MindStudio重启后该CANN处于激活状态。

ii. 若在切换CANN之前，已经有工程打开，则还会弹出如图13-40所示界面，单击“OK”，在随后弹出的如图13-41所示界面单击“Change”，选择切换后的Toolkit版本。

图 13-40 CANN 未安装提示信息

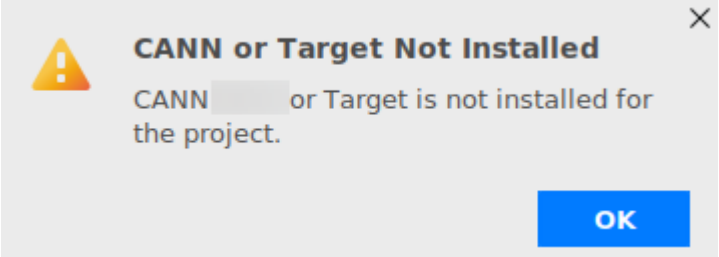
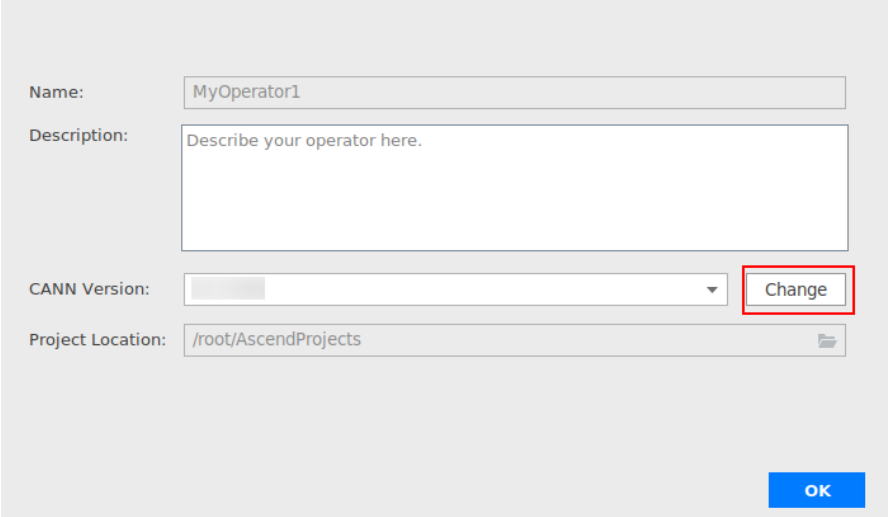


图 13-41 修改工程属性（示例）



iii. 选择“Cancel”，CANN切换到新版本，但是状态为未激活状态，此时新版本CANN不可用。

- **Windows系统：**

  a. 请参考“13.6.1 概述”进入CANN管理页面，单击页面右侧的“Change CANN”，弹出切换CANN界面。

  b. 选中需要替换的Toolkit版本，单击“OK”。弹出确认页面，如图13-42所示。

图 13-42 确认页面

选择“OK”，CANN切换到新版本，MindStudio重启后该CANN处于激活状态。若在切换CANN之前，已经有工程打开，则重启后还会弹出图13-43、图13-44所示界面，用于确认切换后的CANN版本信息。

图 13-43 CANN 未安装提示信息

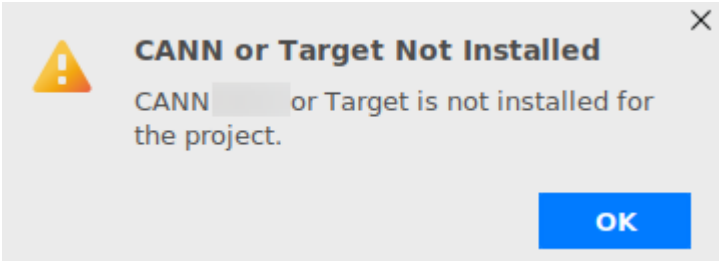
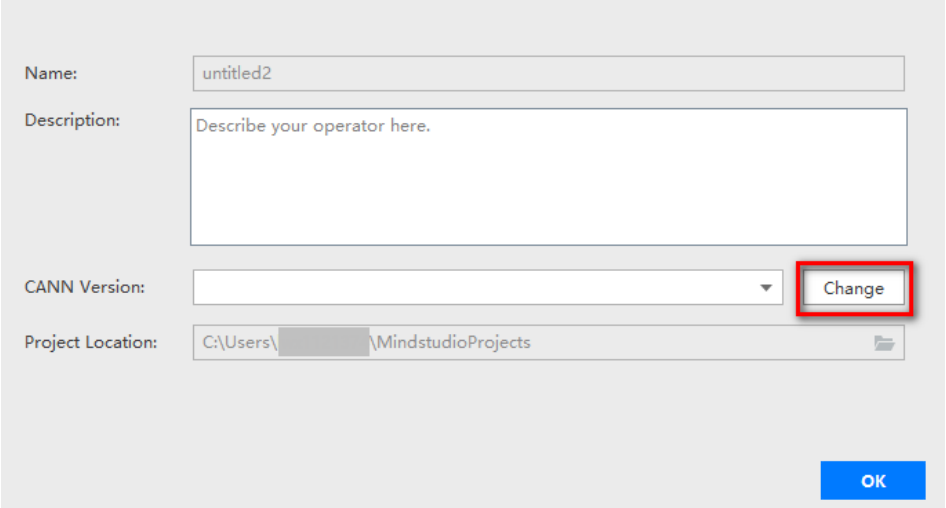


图 13-44 修改工程属性（示例）



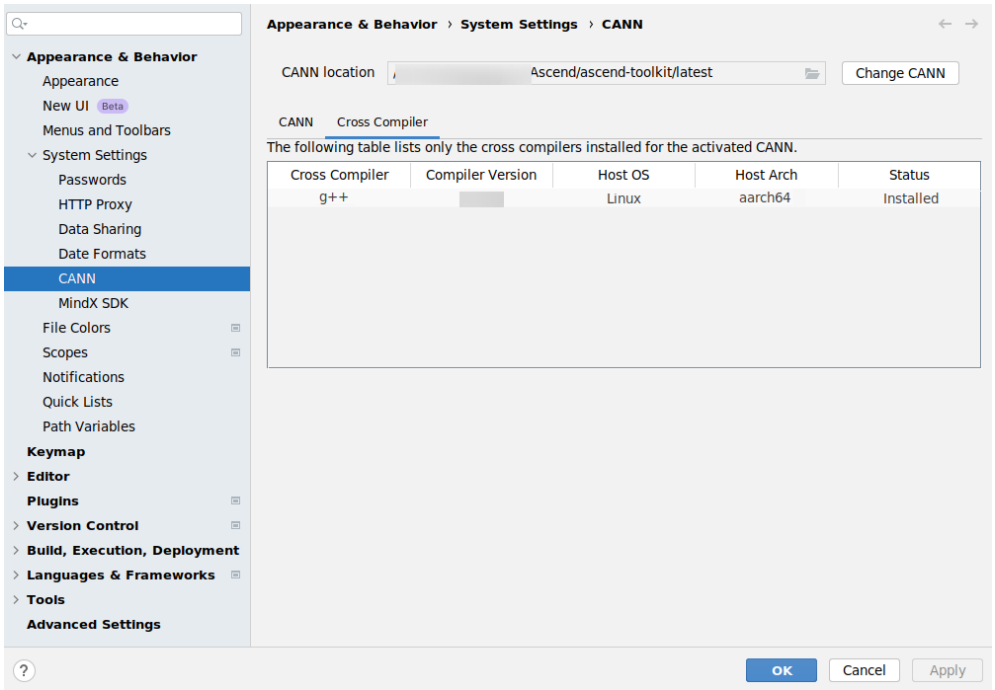
13.6.3 配置交叉编译环境

进入CANN管理页面，在CANN管理界面选择“Cross Compiler”页签，系统显示当前激活状态的CANN需要安装的编译环境，如图13-45所示。如果Status显示“Not installed”，请参见《MindStudio安装指南》的“配置编译环境”章节进行手动配置。

说明

- MindStudio安装服务器环境下的Ascend-cann-toolkit开发套件包版本需要和运行环境下的CANN软件包版本保持一致。
- MindStudio安装服务器环境下需要同时安装Ascend-cann-toolkit开发套件包和运行环境对应的CANN软件包。

图 13-45 配置交叉编译环境



配置完成后，重新启动MindStudio，再次进入CANN管理，可以看到“Status”变成“Installed”。

13.7 日志管理

13.7.1 使用前必读

工具简介

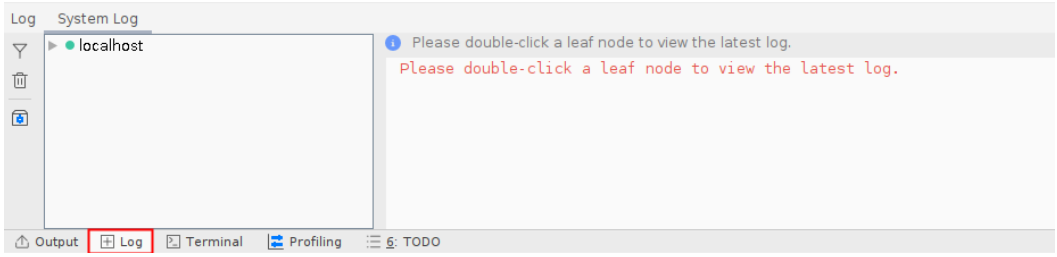
MindStudio为昇腾AI处理器提供覆盖全系统的日志收集与日志分析解决方案，提升运行时算法问题定位效率。MindStudio提供全系统统一的日志格式，并以图形化的形式提供跨平台日志可视化分析能力及运行时诊断能力，提升日志分析系统的易用性。

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13-46所示。

图 13-46 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Windows系统下，使用SSH连接时，目标地址不能是Windows本身的服务器地址。
- 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、Centos7.6-x86/aarch系统和银河麒麟OS V10 SP1-aarch系统：**sudo yum install nautilus gedit**。
  - Centos8.2-x86/aarch系统：**sudo dnf install nautilus gedit**。
- log文件默认使用gedit软件打开，配置命令为**xdg-mime default org.gnome.gedit.desktop text/x-log**。

13.7.2 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：请使用连接到的SSH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。
- 对于已安装芯片的Windows服务器，日志落盘路径为C:\Program Files\Huawei\Ascend\alog\device-{device-id}，需登录此服务器才能对日志进行管理操作。

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 13-47 日志列表（RC 场景下）

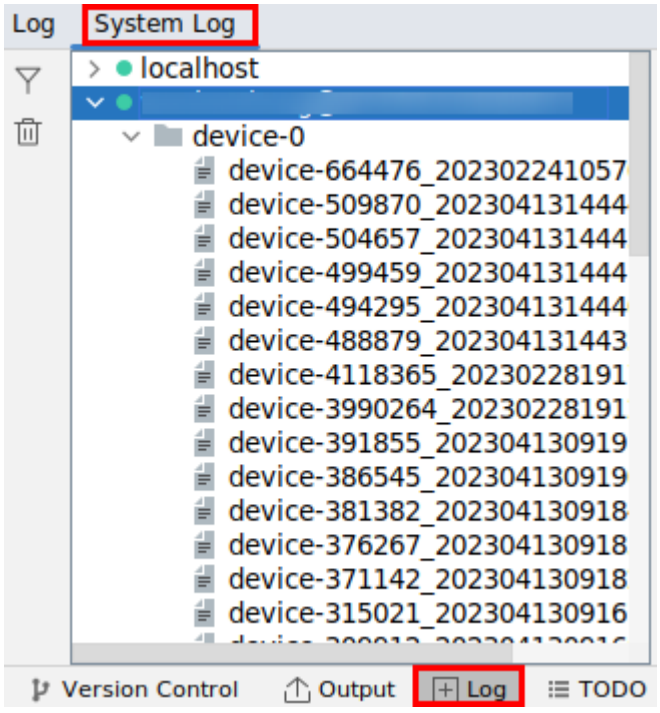
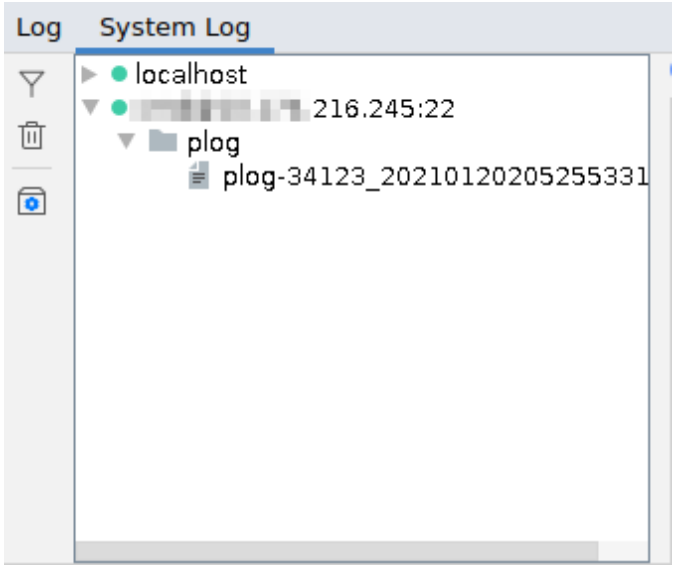


图 13-48 日志列表（EP 场景下）



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 13-49 System log 页签（RC 场景下）

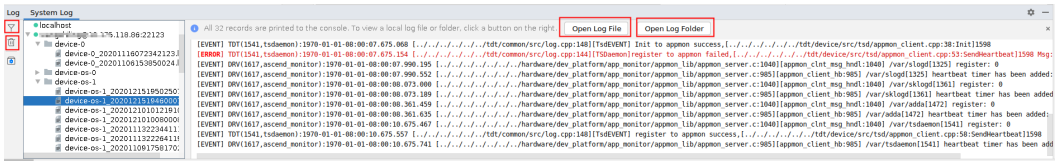
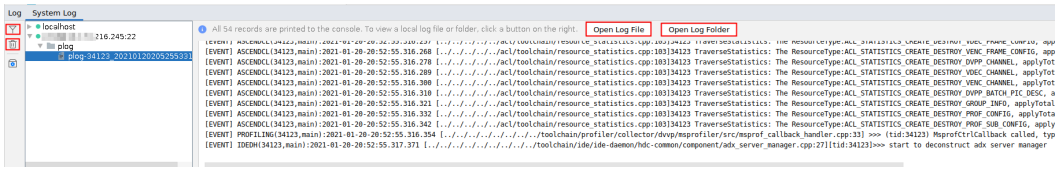


图 13-50 System log 页签（EP 场景下）



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。

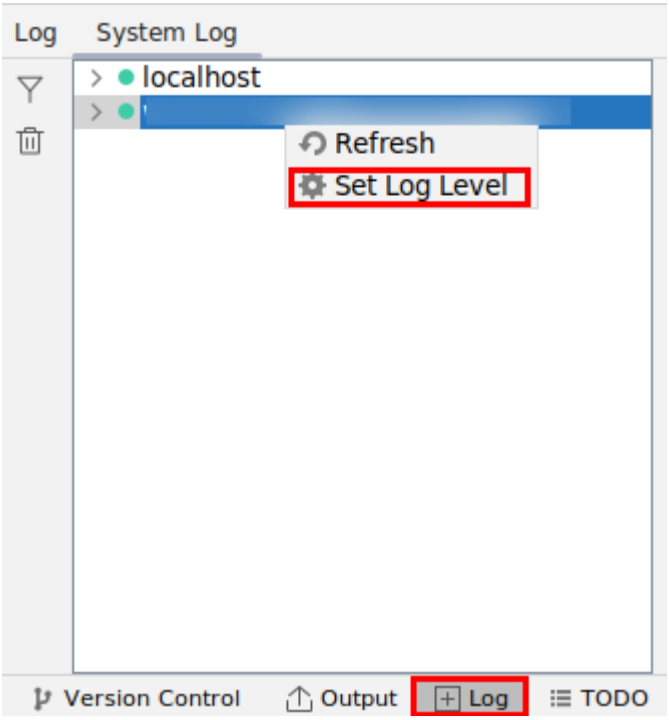
----结束

13.7.3 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界面下方的“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13-51所示。

图 13-51 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13-52所示。

说明

- 日志级别说明请参考《**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 13-52 Set Log Level 窗口

步骤3 设置日志级别后，单击“OK”。

----结束

13.8 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 13-11 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
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）

快捷键	说明
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）

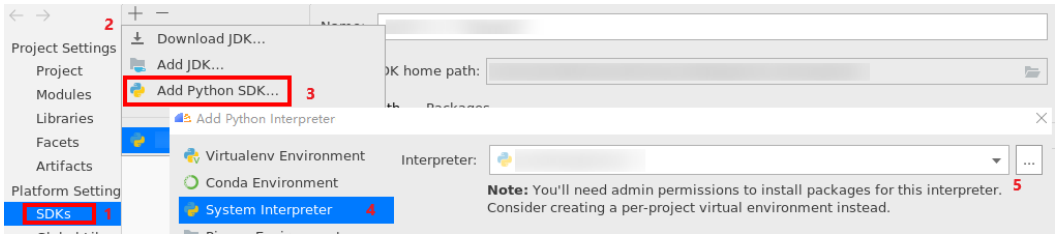
13.9 Python SDK 设置

设置全局 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”，在右侧配置窗口单击“+”，选择“Add Python SDK...”进入Python解析器添加窗口。

图 13-53 添加全局 Python 库



步骤3 在“Add Python Interpreter”窗口中，根据实际情况，选择合适的Python解析器。可选解析器介绍请参见表13-12。单击“OK”添加完成。

图 13-54 Add Python Interpreter

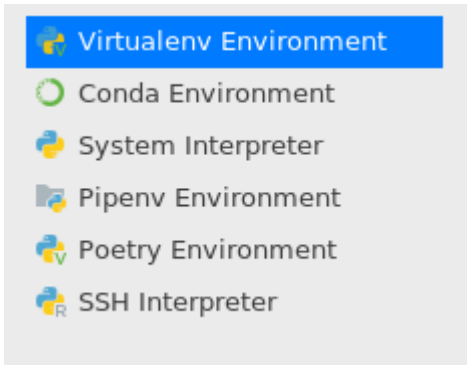


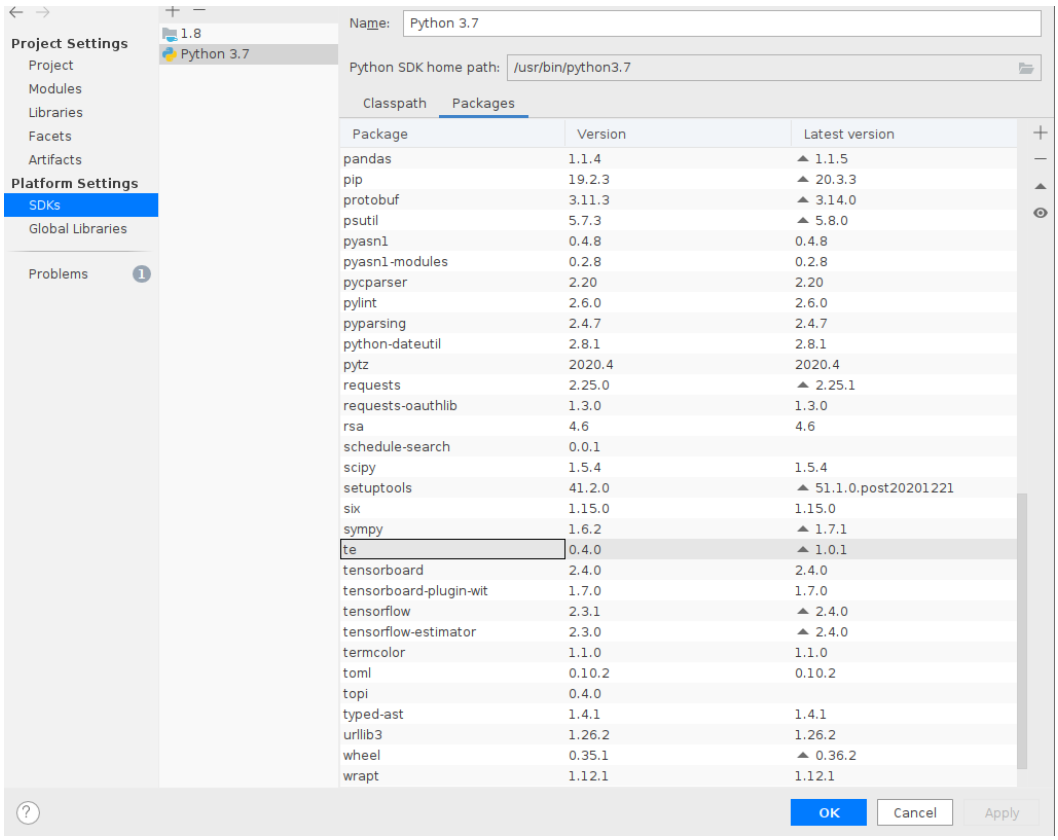
表 13-12 Python 解析器选项

解析器	说明
Virtualenv Environment	集成了Virtual Environment工具，用以创建独立的虚拟环境。

解析器	说明
Conda Environment	使用Anaconda中带有的Python解释器。
System Interpreter	系统本地解析器。
Pipenv Environment	Pipenv是一种工具，提供了为Python项目创建虚拟环境所需的所有必要方法。
Poetry Environment	使用Poetry工具，创建基于工程依赖的虚拟环境。
SSH Interpreter	**远程解析器**。通过SSH方式添加远程设备的Python SDK并通过Deployment功能，将指定项目中的文件、文件夹同步到远程指定机器的指定目录，详细请参见13.3 Ascend Deployment。（如选择该选项，在工程运行时，MindStudio将以远端运行的方式进行。）

步骤4 添加完成后，用户可根据实际情况，自行引入了其他第三方Python库或自定义Python库。单击“OK”，关闭Project Structure窗口。

图 13-55 查看 Python 库



----结束

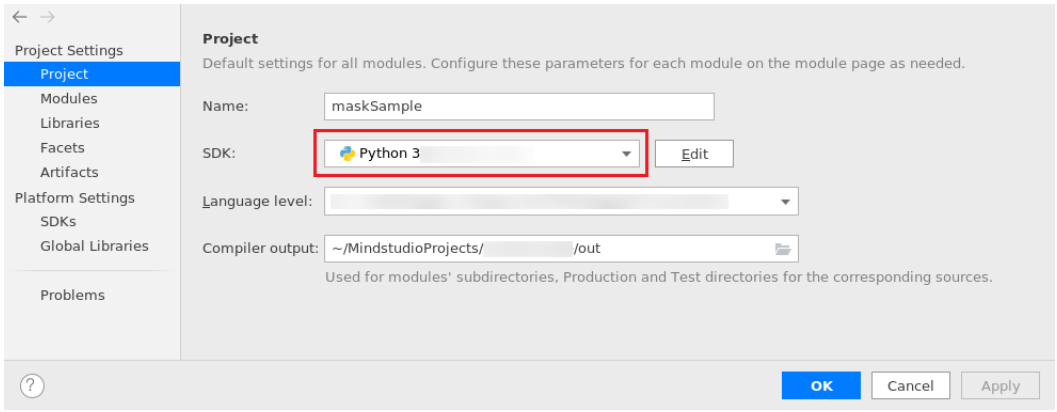
设置工程级 Python SDK

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”，进入“Project”设置。

在“SDK”选项中，下拉并选择已配置的Python SDK或选择“Add Python SDK...”，进入Python解析器添加窗口新增Python SDK，单击“Apply”应用配置，如图13-56所示。

图 13-56 选择 Project SDK



----结束

设置模块级 Python SDK（昇腾工程）

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 > Modules”，进入“Modules”设置。

在新增的“Python”项右侧，下拉选择已配置的Python SDK或单击  自定义Python SDK，单击“Apply”应用配置，如图13-57所示。

图 13-57 添加 Python SDK

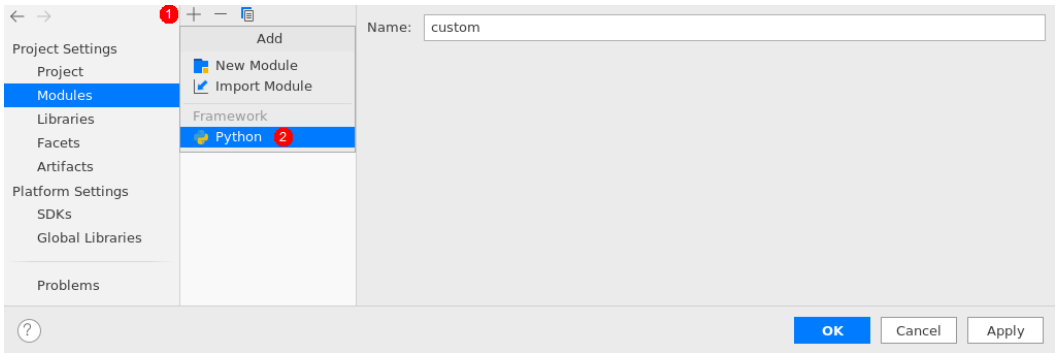
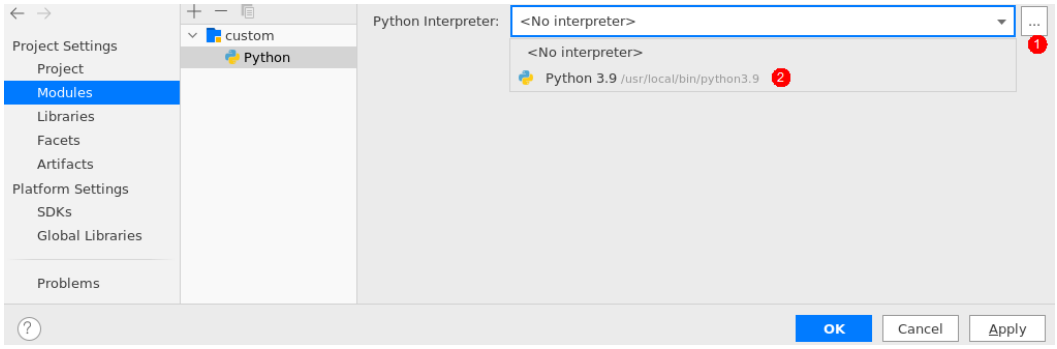


图 13-58 选择已有的 Python SDK 或自定义



----结束

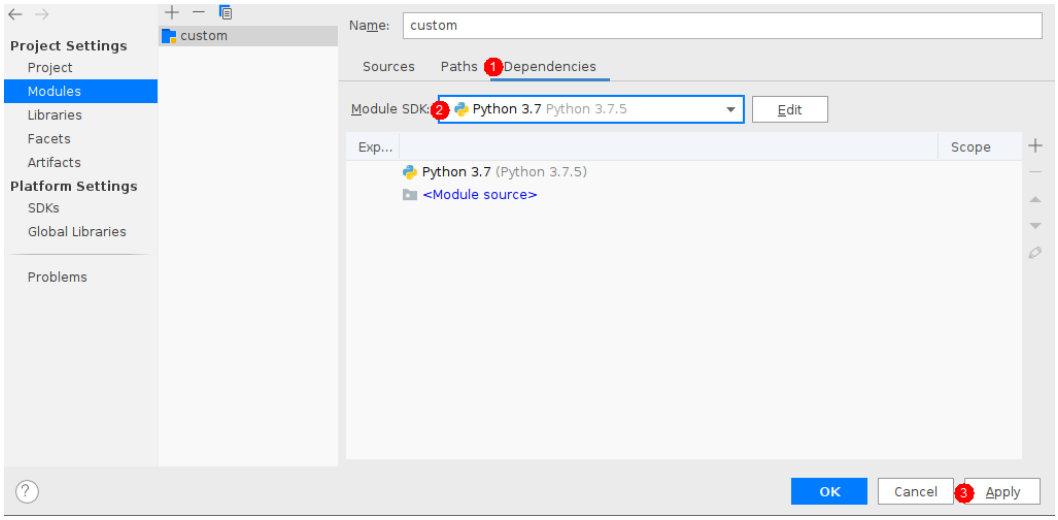
设置模块级 Python SDK（非昇腾工程）

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 > Modules”，进入“Modules”设置。

在“Dependencies”页签中，下拉选择已配置的Python SDK或选择“Add Python SDK...”，进入Python解析器添加窗口新增Python SDK，单击“Apply”应用配置，如图13-59所示。

图 13-59 选择 Modules SDK



----结束

14 附录

14.1 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，远端路径尽量选择个人用户路径。如果对ONNX算子编译时出现异常，需要自行将依赖包google/protobuf目录权限值设置为755。
- 尽可能依赖本用户安装的CANN包；或者至少保证安装CANN包的普通用户是可信任的。以避免提权风险。
- 若使用root启动MindStudio，在配置页面选择普通用户路径下的文件，或者将环境变量配置至普通用户路径下，存在提权风险，所以建议优先使用普通用户启动MindStudio。
- 经由MindStudio调用第三方组件生成的文件、日志等，请用户自行保证权限最小化。
- MindStudio为开发态工具，不建议通过X协议转发进行使用，建议在本地启动使用。
- 若通过HTTPS访问外部网站，请设置浏览器的TLS协议版本为TLS 1.2或者TLS 1.3。
- 使用MindStudio打开、显示、运行的资源（如：模型文件、训练脚本、可执行程序等），需要确定来源可信。对于互联网上来源不可信的资源，请不要使用MindStudio打开，否则有可能存在安全风险。

14.2 公网 URL 及邮箱

MindStudio 6.0.0 公网URL及邮箱.xlsx

14.3 FAQ

14.3.1 获取帮助和提出建议

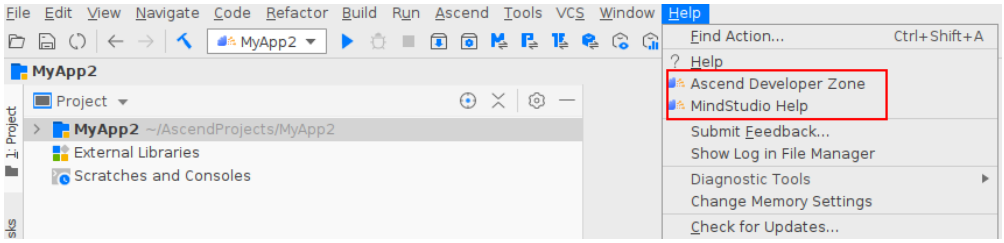
在使用MindStudio的过程中，如果想查看更多相关信息，请直接在MindStudio主界面的菜单栏中单击“Help > MindStudio Help”，该操作会连接网络并跳转至昇腾社区MindStudio（https://www.hiascend.com/software/mindstudio）查看。

如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”，该操作会连接网络并跳转至昇腾论坛（https://bbs.huaweicloud.com/forum/forum-945-1.html）进行反馈。

说明

使用**MindStudio Help**和**Ascend Developer Zone**功能时需要使用浏览器登录网页，如果未安装浏览器，请用户自行安装。

图 14-1 更多信息和帮助



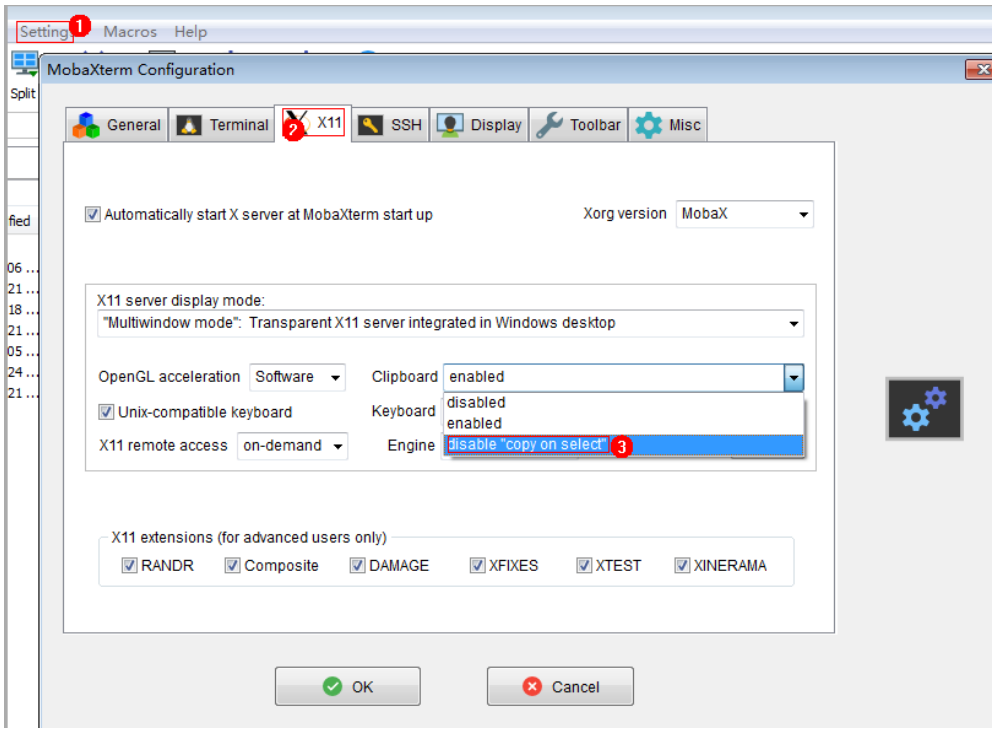
14.3.2 Windows 上远程打开 MindStudio 时，复制的内容无法粘贴到编辑器窗口中

问题现象

Windows上远程打开MindStudio时，复制的内容无法粘贴到编辑器窗口中。

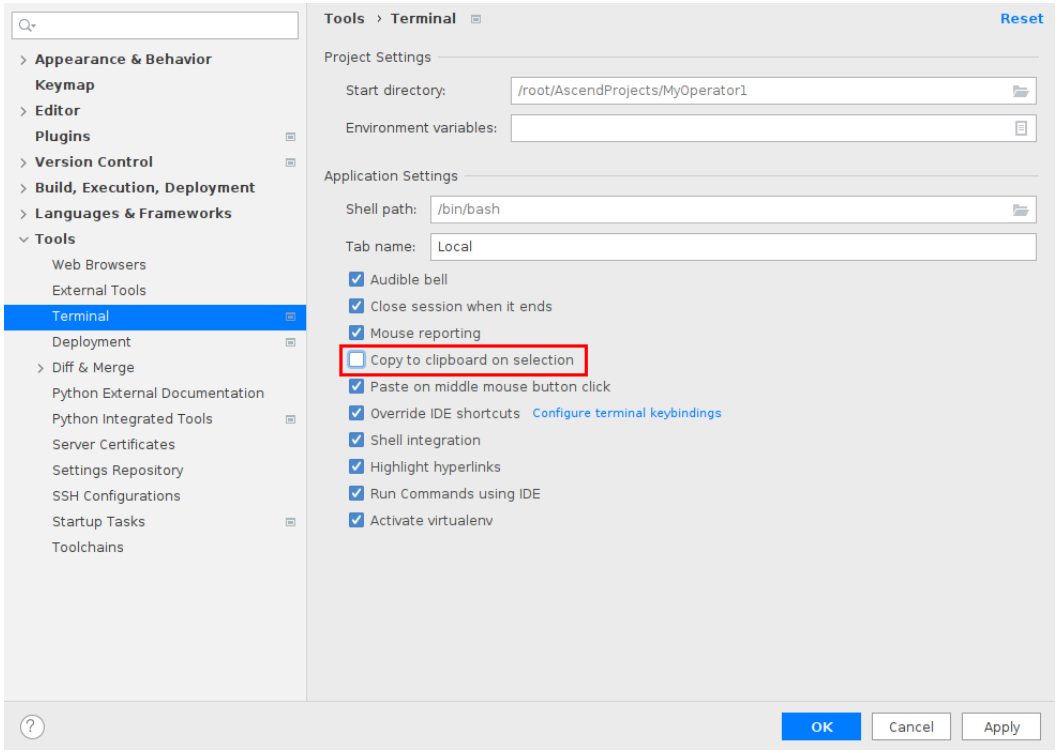
解决方案

步骤1 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。



步骤2 打开MindStudio的“Settings”页面，选择“Tools > Terminal”，取消勾选“Copy to clipboard on selection”选项，如图14-2所示。

图 14-2 MindStudio 设置



----结束

> **说明**
>
> 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。

14.3.3 MobaXterm 启动 MindStudio 编辑字符时，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，方向键变成2468的数字。

解决方案

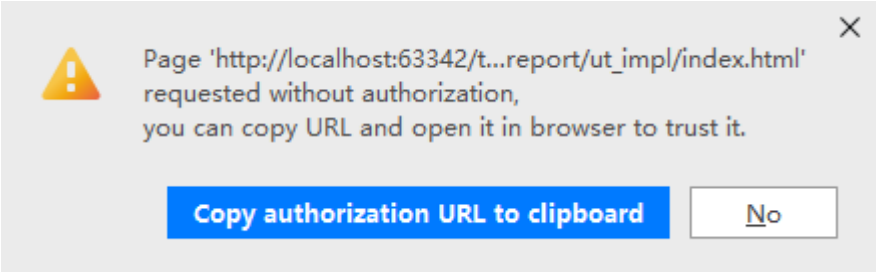
卸载或者切换不使用搜狗输入法，然后用MobaXterm远程连接启动MindStudio。

14.3.4 配置不受信任的网址访问浏览器

问题现象

UT测试用例运行结束后，如果需要通过浏览器打开覆盖率html文件，可能会出现图14-3所示提示。

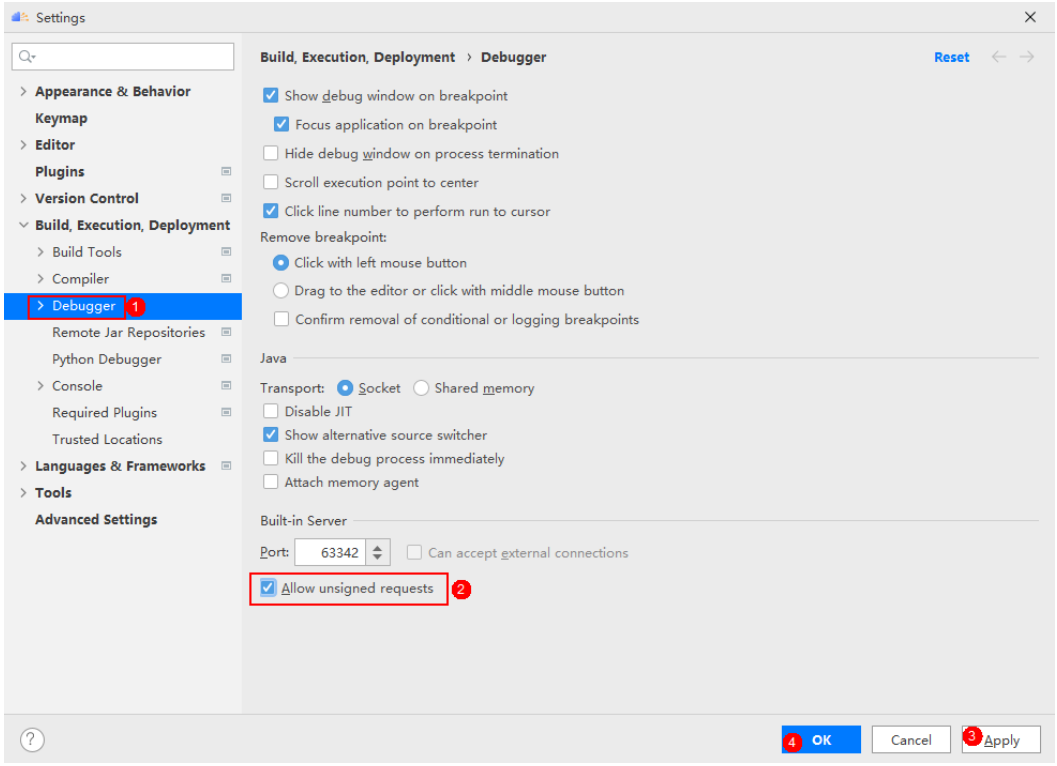
图 14-3 提示



解决方案

请单击菜单栏中的“File > Settings”，参考图14-4修改。

图 14-4 勾选 Allow unsigned requests

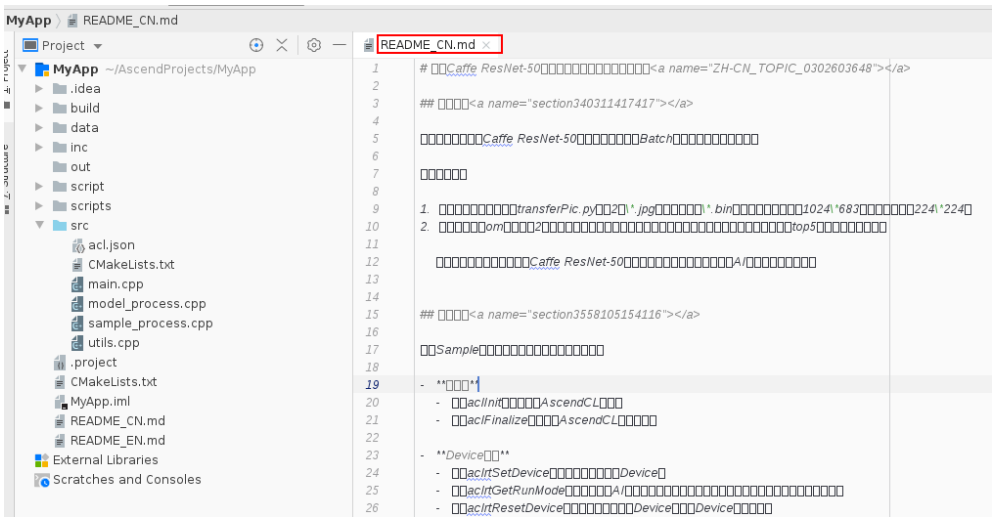


14.3.5 中文显示乱码和界面显示不全不规整

问题现象

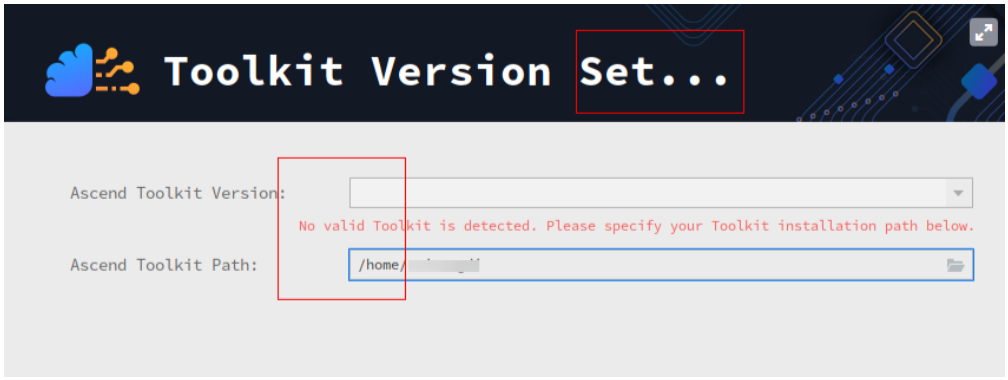
- 例如在新建APP应用工程时，中文显示乱码，例如README_CN.md文件，如图14-5所示。

图 14-5 中文显示乱码



- 界面显示不全不规整，例如标题显示不全，如图14-6所示。

图 14-6 标题显示不全



解决方案

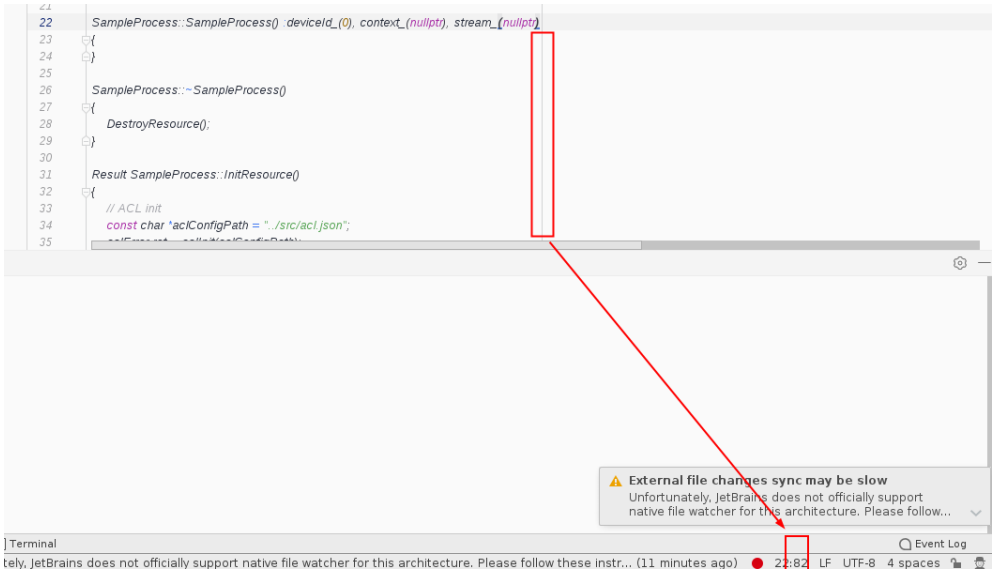
请自行安装微软雅黑字体(msyh.ttc)。

14.3.6 代码最大行宽与参考值不一致

问题现象

打开工程目录下的文件时，代码最大行宽与参考值不一致，参考值一般默认为120。如图14-7所示。

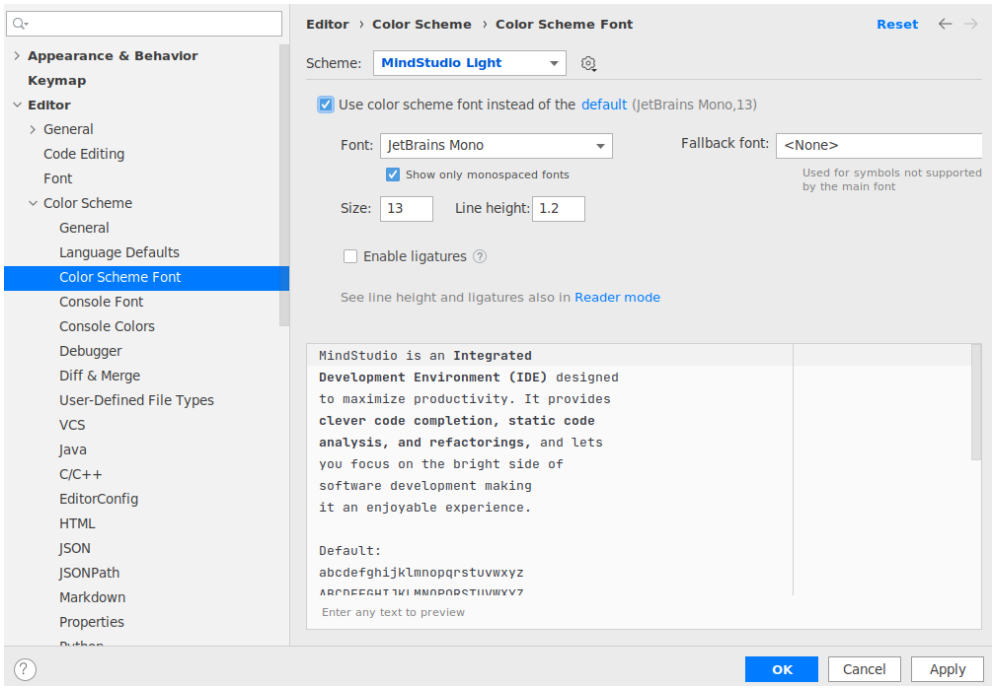
图 14-7 代码最大行宽与参考值不一致



解决方案

步骤1 在MindStudio主界面选择“File > Settings > Editor > Color Scheme > Color Scheme Font”，进入字体选择界面，如图14-8所示。

图 14-8 Color Scheme Font



步骤2 在“Font”参数后的下拉框中选择与当前字体不一致的字体，单击“Apply > OK”，返回工程目录下的文件查看代码最大行宽与参考值是否一致。

- 是，操作完成。
- 否，重复进行步骤**步骤1**~**步骤2**直至代码最大行宽与参考值一致。

说明

如果在等宽字体中未能找到合适字体，请把“Show only monospaced fonts”参数前的小方框去掉勾选，在等宽字体除外的其他字体中进行选择。

----结束

14.3.7 MindStudio 异常退出导致文件丢失

问题现象

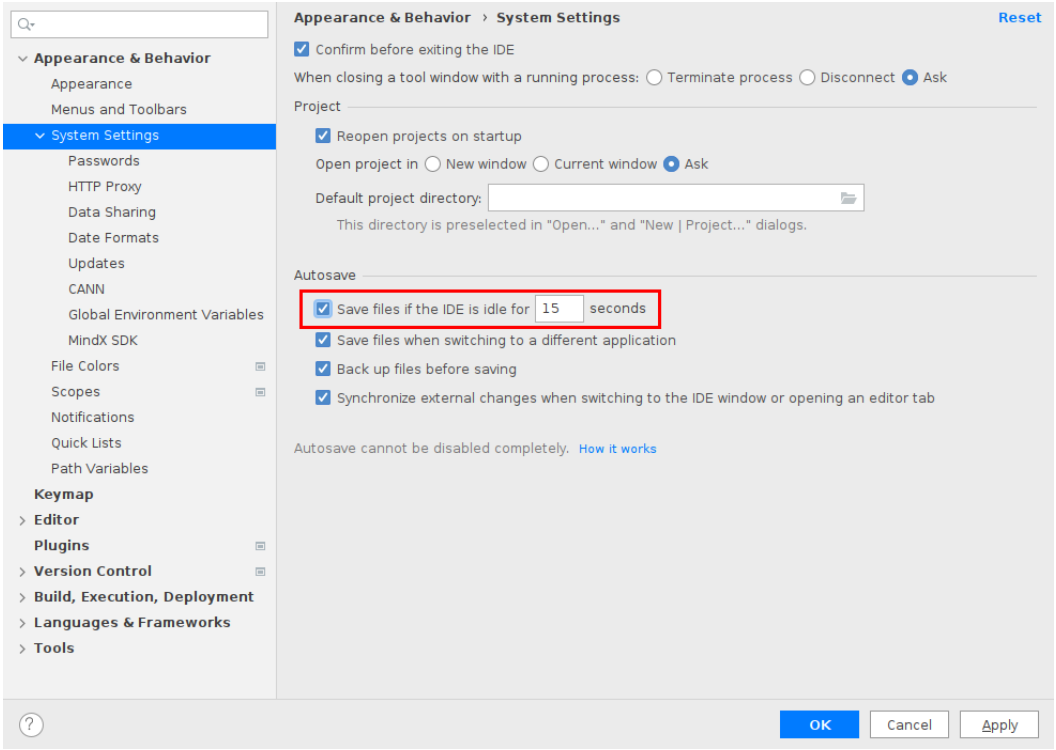
MindStudio异常退出时，文件因未即使保存而导致丢失。

解决方案

通过以下设置，可实现MindStudio空闲15s后自动保存文件，避免因异常情况导致文件未保存而丢失。

打开MindStudio的“Settings”页面，选择“Appearance & Behavior > System Settings”，勾选“Save files if the IDE is idle for *xx* seconds”选项，如**图14-9**所示。可根据个人情况设置具体时间。

图 14-9 自动保存

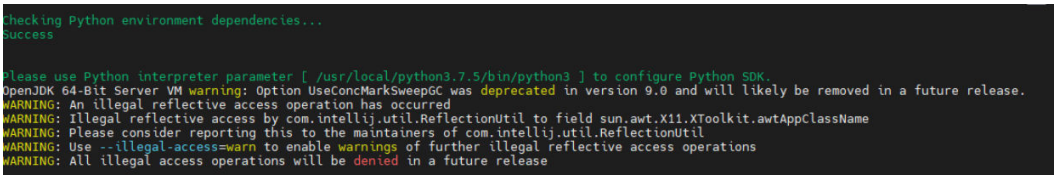


14.3.8 MindStudio 界面闪退后，重新启动 MindStudio 时提示进程已存在

问题描述

由于网络异常等原因导致MindStudio界面闪退。用户重新启动MindStudio，报错提示已有进程存在。如图14-10所示。

图 14-10 提示信息



解决方案

步骤1 执行**ps -ef | grep java**命令查看后台进程。找到对应的进程号PID，例如713。

步骤2 执行**kill -9**命令关闭已有进程。例如执行**kill -9 713**命令关闭713进程。

步骤3 执行**./MindStudio.sh**重新启动MindStudio。

----结束

14.3.9 C 语言 printf 信息在 Debug 时 Console 中概率性没有内容输出

问题现象

C语言printf信息在Debug时Console中概率性没有内容输出。

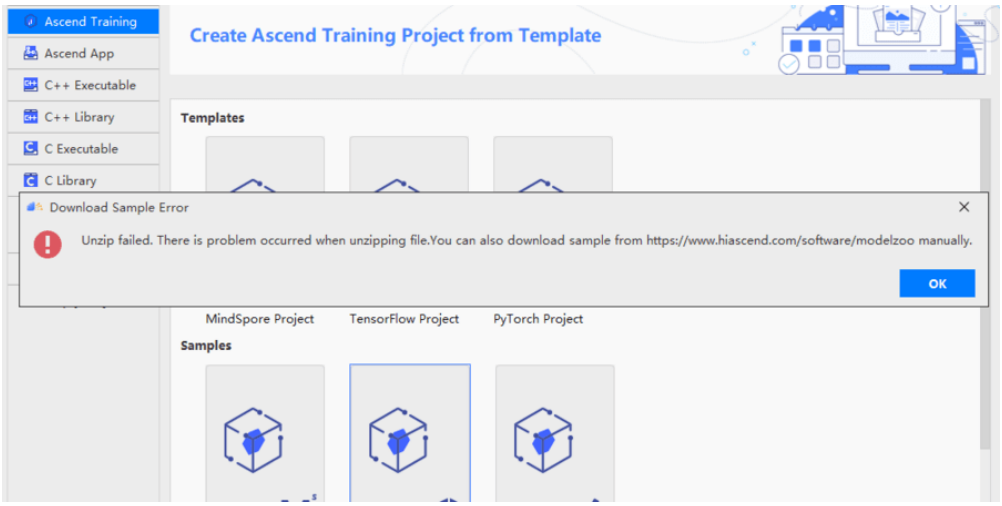
解决方案

这是gdb内部的bug，可以通过setbuf或setvbuf设置无缓冲，或者printf后主动调用fflush及时输出缓冲来规避该问题。

14.3.10 在 Windows 系统环境下创建训练样例工程时报错“Unzip failed. There is probleam occurred when unzipping file.”

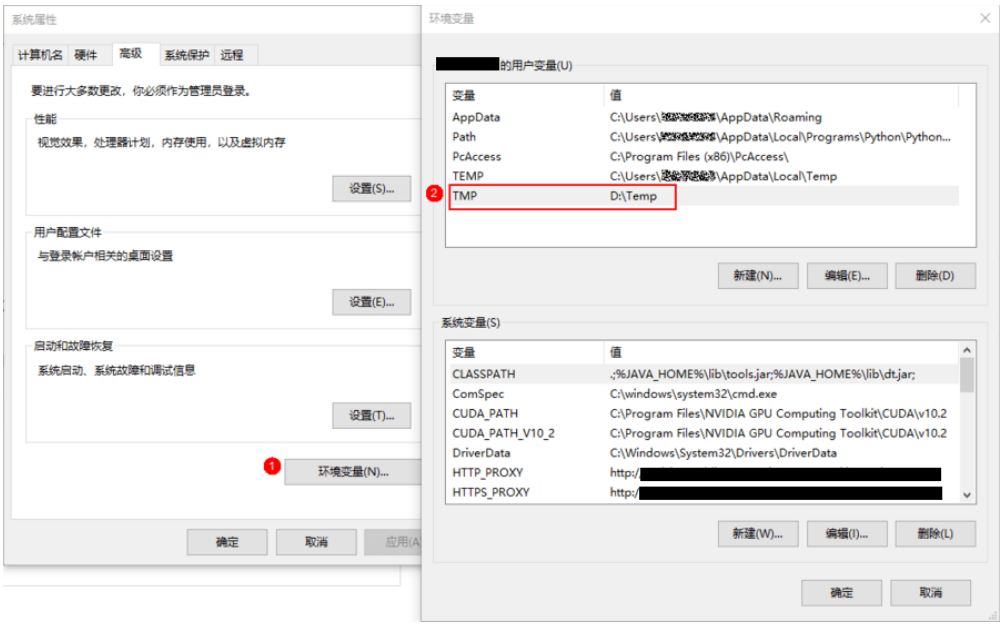
问题现象

在Windows系统环境下创建训练样例工程时，弹出报错对话框，提示“Unzip failed. There is probleam occurred when unzipping file.”



解决方案

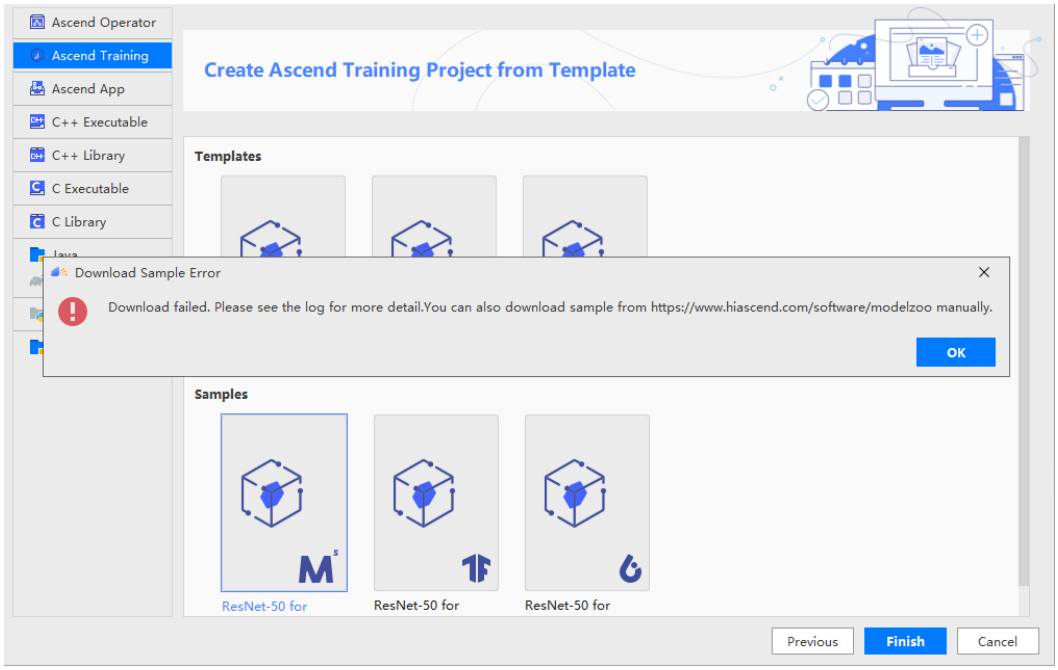
修改环境变量TMP，指向非用户目录下的其他有读写权限的目录，如D:\temp。



14.3.11 创建训练样例工程时报错“Download failed. Please see the log for more detail. You can also download sample from https://www.hiascend.com/software/modelzoo manually.”

问题现象

创建训练样例工程时，弹出报错对话框，提示“Download failed. Please see the log for more detail. You can also download sample from https://www.hiascend.com/software/modelzoo manually.”



解决方案

1. 检查网络是否为连通状态，若网络不通则先将网络调试为连通状态。
2. 若网络为连通状态，则查看日志，若日志出现如下图中字段“No module named 'requests'”，则需要进入命令行界面执行命令“**pip3 install requests**”安装requests模块。

```
    at com.intellij.openapi.updateSettings.impl.UpdateChecker.access$doUpdateAndShowResult(UpdateChecker.kt:56)
    at com.intellij.openapi.updateSettings.impl.UpdateChecker$updateAndShowResult$1.run(UpdateChecker.kt:108)
    at com.intellij.util.RunnableCallable.call(RunnableCallable.java:20)
    at com.intellij.util.RunnableCallable.call(RunnableCallable.java:11)
    at com.intellij.openapi.application.impl.ApplicationImpl$1.call(ApplicationImpl.java:265)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.util.concurrent.Executors$PrivilegedThreadFactory$1$1.run(Executors.java:668)
    at java.base/java.util.concurrent.Executors$PrivilegedThreadFactory$1$1.run(Executors.java:665)
    at java.base/java.security.AccessController.doPrivileged(Native Method)
    at java.base/java.util.concurrent.Executors$PrivilegedThreadFactory$1.run(Executors.java:665)
    at java.base/java.lang.Thread.run(Thread.java:834)
2022-03-08 10:50:19,512 [  13559]   INFO - ateSettings.impl.UpdateChecker - Connection failed (connect timed out). Please chec
2022-03-08 10:50:19,585 [  13632]   INFO - ge.utils.TrainingSampleManager - Download training sample MINDSPORE_RESNET50_SAMPLE
2022-03-08 10:50:19,655 [  13702]   WARN - utils.TrainingSampleDownloader - Download failed. Traceback (most recent call last)
  File "E:\mindstudio\MindStudio_5.0.T101_win\plugins\ascend-foundation\tools\download_file.py", line 11, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
2022-03-08 10:50:19,658 [  13705]   WARN - d.DTrainingWizardModuleBuilder - Download failed. Please see the log for more detai
2022-03-08 10:50:20,041 [  14088]   WARN - arketplace.MarketplaceRequests - Can not get JetBrains plugins' IDs from Marketplac
2022-03-08 10:50:20,270 [  14317]   WARN - arketplace.MarketplaceRequests - Can not get JetBrains plugins' IDs from Marketplac
2022-03-08 10:50:20,774 [  14821]   WARN - arketplace.MarketplaceRequests - Can not get JetBrains plugins' IDs from Marketplac
2022-03-08 10:50:20,880 [  14927]   WARN - arketplace.MarketplaceRequests - Can not get JetBrains plugins' IDs from Marketplac
2022-03-08 10:50:25,076 [  19123]   WARN - arketplace.MarketplaceRequests - Error reading Marketplace file: url=
http://plugins.jetbrains.com/plugins/list/?uuid=c4f066e8-1ff5-4694-8ffc-c5205aa46765&build=MS-211.6693.111.3 file=availables.x
```

日志落盘路径为：

- Linux：/home/XXXX/.cache/Huawei/MindStudioMS-XX/log/idea.log
- Windows：C:\Users\XXX\AppData\Local\Huawei\MindStudioMS-XX\log\idea.log

14.3.12 设置超大内存后无法启动 MindStudio

问题描述

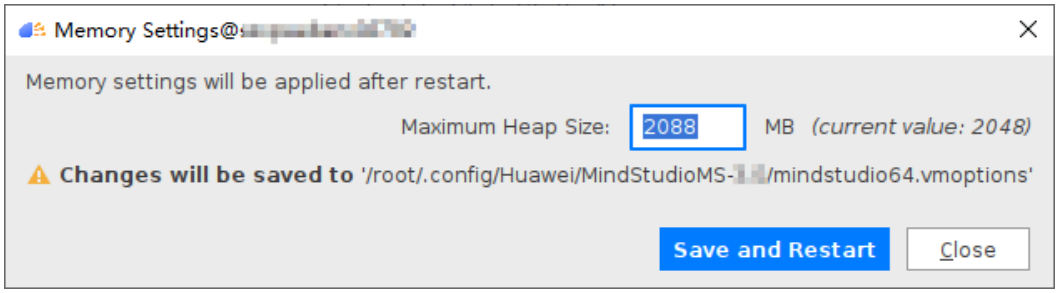
- 在Windows启动MindStudio失败。
- 在Linux启动MindStudio时，报错“Cound not reserve enough space for xxx object heap”，提示信息如下。

```
root@[redacted]:~/[redacted]/MindStudio/bin# ./MindStudio.sh
Error occurred during initialization of VM
Could not reserve enough space for 2097152000000KB object heap
```

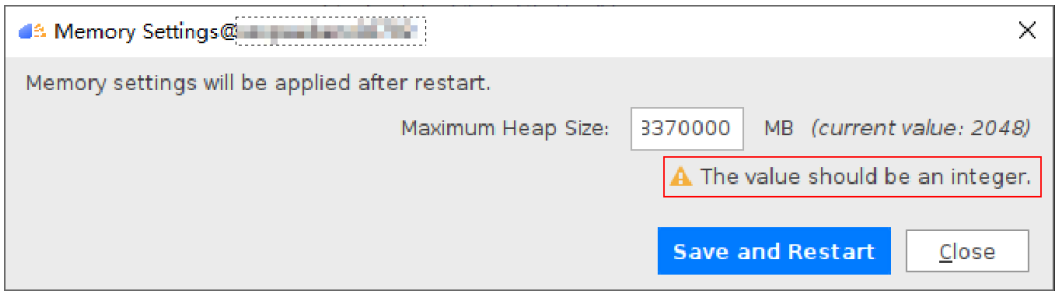
问题原因

用户在菜单栏选择“Help > Change Memory Settings”，弹出的Memory Settings窗口（以图14-11为例）中设置的运行内存大小超过启动运行主机的内存大小。如主机内存为512MB，而用户设置内存为2014MB。

图 14-11 Memory Settings（Linux）



设置的运行内存大小应在256~2147483647范围内。若超过该范围，单击“Save and Restart”，将提示“The value should be an integer”。



解决方法

访问用户目录下的mindstudio64.exe.vmoptions文件，修改-Xmx参数的值，设置一个合理的大小，该值至少小于主机内存。

- Windows端：C:\Users*个人用户*\AppData\Roaming\Huawei\MindStudioMS-*{version}*\mindstudio64.vmoptions
- Linux端： ~/.config/Huawei/MindStudioMS-*{version}*/mindstudio64.vmoptions

```
-Xms128m
-Xmx2048000000m
-XX:ReservedCodeCacheSize=512m
-XX:+UseG1GC
-XX:SoftRefLRUPolicyMSPerMB=50
-XX:CICompilerCount=2
-XX:+HeapDumpOnOutOfMemoryError
-XX:-OmitStackTraceInFastThrow
-ea
-Dsun.io.useCanonCaches=false
-Djdk.http.auth.tunneling.disabledSchemes=""
-Djdk.attach.allowAttachSelf=true
-Djdk.module.illegalAccess.silent=true
-Dkotlinx.coroutines.debug=off
-Dsun.tools.attach.tmp.only=true
~
```

14.3.13 在 Linux 端使用 MindStudio 时出现阻塞，等待不定时间方可恢复

问题描述

- 启动MindStudio时出现界面无响应，且控制台阻塞在日志输出“WARN - o.ssh.connection.utils.KMCUtil - start init kmc”。

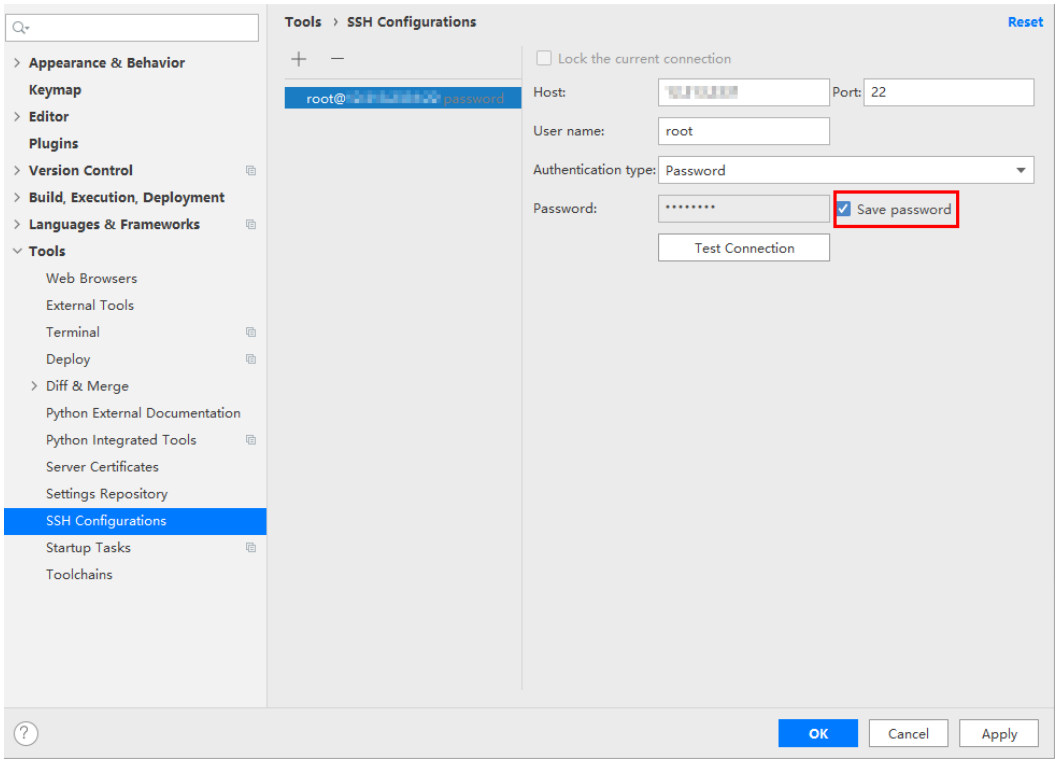
- 使用MindStudio时出现阻塞，且查看/root/.cache/Huawei/MindStudioMS-x.x/log/idea.log，有如下日志报错“WARN - o.ssh.connection.utils.KMCUtil - kmc warn entropy may not enough, entropy_avil=2, read_wakeup_threshold=64”。

问题原因

曾在如图14-12SSH连接配置界面设置保存密码。SSH插件使用KMC加密套件做密码保存。而/dev/random产生随机数的原理是利用当前系统的熵池（random pool）来计算

出固定一定数量的随机比特，然后将这些比特作为字节流返回。熵池就是当前系统的环境噪音，熵指的是一个系统的混乱程度，系统噪音可以通过很多参数来评估，如内存的使用，文件的使用量，不同类型的进程数量等等。如果当前环境噪音变化的不是很剧烈或者当前环境噪音很小，比如设备刚启动时产生的随机数熵值不足，/dev/random会阻塞当前的程序，直到根据熵池产生新的随机字节之后才返回。当熵源不足时，使用haveged组件进行补熵。

图 14-12 SSH 连接



解决方案

通过添加haveged服务解决随机数阻塞问题。

步骤1 检查操作系统是否安装haveged服务。

- 对于CentOS、OpenEuler、EulerOS 、Kylin系统，执行以下命令进行检查。
  ```
  /bin/rpm -qa | grep -w "haveged"
  ```
- 对于Ubuntu系统，执行以下命令进行检查。
  ```
  dpkg -l | grep -w "haveged"
  ```

若没有回显数据，则表示未安装haveged服务。

步骤2 安装haveged服务。

- 对于CentOS、OpenEuler、EulerOS 、Kylin系统，执行以下命令进行安装。
  ```
  sudo yum install haveged
  ```
- 对于Ubuntu系统，执行以下命令进行安装。
  ```
  sudo apt-get install haveged
  ```

步骤3 检查服务是否已经启动。

```
systemctl status haveged.service
```

- 回显包含“Active: active (running)”信息，表示服务已启动。

- 回显包含“Active: inactive (dead)”信息，表示服务未启动，执行以下命令启动服务。
  ```
  systemctl start haveged.service
  ```

步骤4 设置该服务开机自启动。

```
systemctl enable haveged.service
```

----结束

14.3.14 基于 MindSDK 开发应用，运行应用工程时，出现“can not find the element factory : mxpi_*xxx*postprocessor”

问题描述

使用MindSDK时，出现“can not find the element factory : mxpi_*xxx*postprocessor”，先在“*$HOME*/Ascend/mindx_sdk/*<sdk_version_package>*/opensource/bin”路径下执行./gst-inspect-1.0 mxpi_*xxx*postprocessor(插件名) 检查插件，发现插件能够正常加载，但运行时仍然报同样的错误信息。

原因分析

gstreamer的历史缓存没有清除。

解决方案

步骤1 确认环境已安装Python3.9，将“libpython3.9.so.1.0”拷贝至/usr/lib64/路径下。

步骤2 执行以下命令（根据具体运行环境选择x86_64.bin或者aarch64.bin）清除gstreamer的历史缓存，再运行程序即可。

```
rm ~/.cache/gstreamer-1.0/registry.x86_64.bin
```

----结束

14.3.15 编译时出现依赖不存在的报错“libgfortran.so.4: No such file or directory”

原因分析

opensource文件中的so文件需要依赖libgfortran.so.4，在编译环境中找不到。

解决方案

执行以下命令安装gfortran（如果环境中找不到gfortran-4，也可以安装大于或等于4的版本）。

```
apt-get update
apt-get install gfortran-4
```


14.3.16 开发昇腾工程过程中，编译配置切换 Toolchain 后编译并运行时报错“cannot execute binary file: Exec format error”

问题描述

在开发昇腾工程过程中，改变编译配置中的Toolchain，切换了编译架构，再次编译并在对应架构的环境中运行，出现如下报错。

```
cannot execute binary file: Exec format error
```

原因分析

工程内容未做任何改动的情况下，切换编译配置改变编译架构，并使用了“Build CMake Project”增量编译，导致最终编译结果未重新生成。

解决方案

工程切换编译配置后，使用“Rebuild CMake Project”进行全量编译，或使用“Clean CMake Project”清理编译结果后，再次使用“Build CMake Project”重新进行编译。

14.3.17 MindStudio 在启动时报错“java.io.IOException: No space left on device”

问题描述

启动MindStudio失败，并出现如下报错。

```
java.io.IOException: No space left on device
```

原因分析

MindStudio长时间运行后，在/home/{username}/tmp目录下会产生较多临时文件。如果MindStudio出错关闭或者由于其他意外原因未能正常关闭，tmp目录下的临时文件可能无法自动删除，从而占据大量磁盘空间，导致再次启动MindStudio时磁盘空间不足。

解决方案

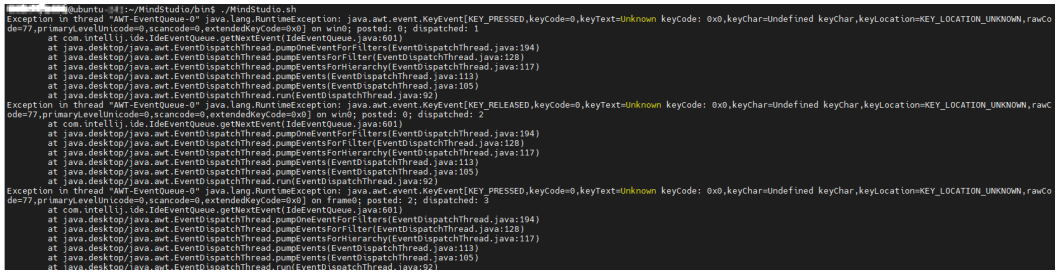
手动清理/home/{username}/tmp目录下的临时文件，释放磁盘空间。

14.3.18 输入框无法输入或删除字符，且后台报错“java.lang.RuntimeException: java.awt.event.KeyEvent“

问题描述

在使用Windows通过另一台Windows远程连接一台Linux设备上的MindStudio时，出现输入框无法输入或删除字符，且后台报“java.lang.RuntimeException: java.awt.event.KeyEvent“错误。

图 14-13 报错信息（示例）



问题分析

由于Windows远程连接和X11转发这两个过程存在键盘间的相互映射，如果这种映射关系不正确，就会导致键盘键入不能够被正确识别。

解决方案

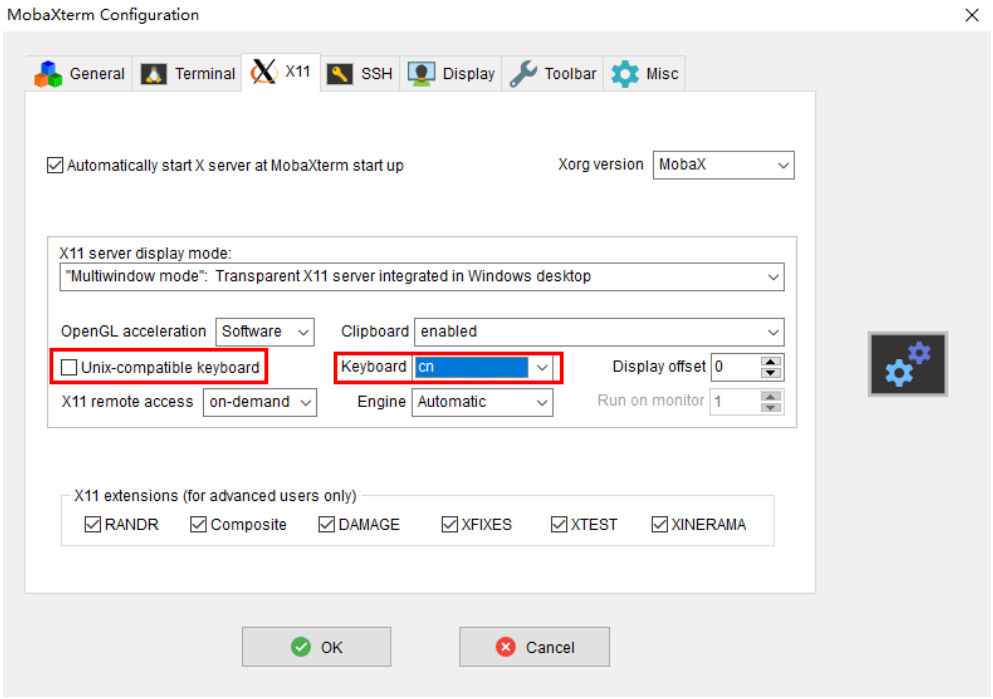
步骤1 请根据Linux系统类型使用以下命令安装X11相关依赖。

- Ubuntu：
  ```
  apt-get install -y xterm xorg x11-apps libxtst-dev libxext-dev libxrender-dev
  ```
- CentOS：
  ```
  yum install -y xorg-x11-apps xterm libXext libXtst libXrender
  ```

步骤2 配置MobaXterm上“X11”项。

1. 在Linux上通过**localectl**命令查看设备的键盘配置，会出现类似如下回显信息。
   ```
   System Locale: LANG=en_US.UTF-8
   LANGUAGE=en_US:en
   VC Keymap: n/a
   X11 Layout: cn
   X11 Model: pc105
   ```
2. 在MobaXterm界面中，单击菜单栏“Settings > Configuration”，在弹出的MobaXTerm配置窗口切换至“X11”页签，取消勾选“Unix-compatible Keyboard”，并设置“KeyBoard”值为**步骤2.1**查询获得的**X11 Layout**值，如**图14-14**所示。

图 14-14 X11 配置界面



3. 单击“OK”，完成配置并重启MindStudio。

----结束

14.3.19 基于 MindSDK 开发应用，使用 MindSDK Pipeline 功能查看可视化 pipeline 时，发现插件置灰不可用

问题描述

在MindStudio中开发基于MindSDK应用，使用MindSDK Pipeline功能查看可视化pipeline时，发现其中部分或全部插件置灰不可用。

问题分析

MindStudio在启动时未获取到插件在库中的文件路径。

解决方案

需确认是否存在自定义插件，根据实际情况选择以下方式进行处理。

- 如当前pipeline未使用自定义插件，重新配置并生效SDK环境变量解决，具体操作可参见**7.2.1 开发前须知**。
- 如存在自定义插件，可通过**MindSDK Pipeline**中的“Plugin Manager”功能，导入自定义插件目录（目录权限需为“440”）。

刷新插件库：打开pipeline并通过插件库中的 按钮刷新，可重新查看到自定义插件已完成加载。

14.3.20 在编写 C++工程时，后台 console 或日志打印出大量 Warning 信息

问题描述

在使用MindStudio编写C++工程时，后台Console界面或日志中打印出大量“AnnotationHolder.createInfoAnnotation()”的Warning日志。

问题分析

因第三方插件处于接口过渡时期，部分接口即将废弃，导致出现大量相关警告。

解决方案

该Warning告警仅提示接口废弃信息，不影响MindStudio的使用，用户可忽略该提示继续编写。

14.3.21 进行 C++工程调试时无法结束

问题描述

使用MindStudio进行C++工程单步调试时无法结束，后台Console界面告警。

问题分析

可能存在依赖版本不匹配或缺失的情况，需根据实际报错信息进行处理。

解决方案

查看Console界面打印的日志，根据其中报错或告警的信息进行处理，安装缺失的依赖或调整依赖的版本。

例如：

- “Missing separate debuginfos...”：则需要安装其中缺失的debuginfos。
- “Warning: Unable to find libthread_db matching inferior's thread library, thread debugging will not be available”：则需重新确认libc与libthread_db软件包版本匹配情况。

14.3.22 执行工程转换将非昇腾工程转换为昇腾工程后，发现参数选择错误无法修改

问题描述

执行工程转换将非昇腾工程转换为昇腾工程后，发现Project Type或Framework/Sub Type参数选择错误，但无法修改和回退。

解决方案

在左侧工程目录中选中新创建的.project文件，单击鼠标右键，在弹出的菜单中单击“Delete...”，按照提示删除文件后，重新选择正确的Project Type和Framework/Sub Type，执行工程转换。

14.3.23 MindStudio 打开工程后，出现操作迟缓、卡顿现象

问题描述

MindStudio打开工程后，操作迟缓、卡顿，比如打开工程缓慢、编辑时卡顿。

问题分析

工程文件下可能存在较大的数据文件。

解决方案

将数据文件从工程目录转移到其他目录下。

14.3.24 MindStudio 在配置远端 SDK 后运行工程，前端界面无法显示动态进度条信息

问题描述

MindStudio在配置远端SDK后运行工程，前端界面无法显示动态进度条信息。

问题分析

MindStudio暂时不支持获取远端的动态变化数据在前端显示。

解决方案

动态进度条只是显示任务的执行进度，不影响工程的正常运行，用户可忽略并等待执行结果。

14.3.25 glibc 版本过低导致无法使用模型可视化功能

问题描述

当Linux环境为aarch64架构时，无法使用MindStudio的模型可视化功能。

问题分析

系统中的glibc版本小于2.29。

解决方案

重新安装2.29版本的glibc，目前只提供了Ubuntu和EulerOS的glibc安装包。

步骤1 请根据操作系统下载对应的安装包，然后上传到MindStudio安装服务器任意路径（比如：*$HOME/package*）并解压。

- Ubuntu：**下载链接**。
- EulerOS：**下载链接**。

步骤2 配置环境变量，在MindStudio安装路径下的bin目录中执行**vi MindStudio.sh**命令，打开MindStudio.sh启动脚本文件，在文件第167行后面添加以下内容（以非root用户为例）。

```
export LD_LIBRARY_PATH=$HOME/package/glibc-2.29/build/math/:$LD_LIBRARY_PATH
export LD_PRELOAD=MindStudio安装目录/jbr/lib/libcef.so:$LD_PRELOAD
export LD_LIBRARY_PATH=MindStudio安装目录/jbr/lib/libcef.so:$LD_LIBRARY_PATH
```

步骤3 执行**:wq!**命令保存文件并退出。

步骤4 在MindStudio安装路径下的bin目录中使用如下命令重启MindStudio。

```
./MindStudio.sh
```

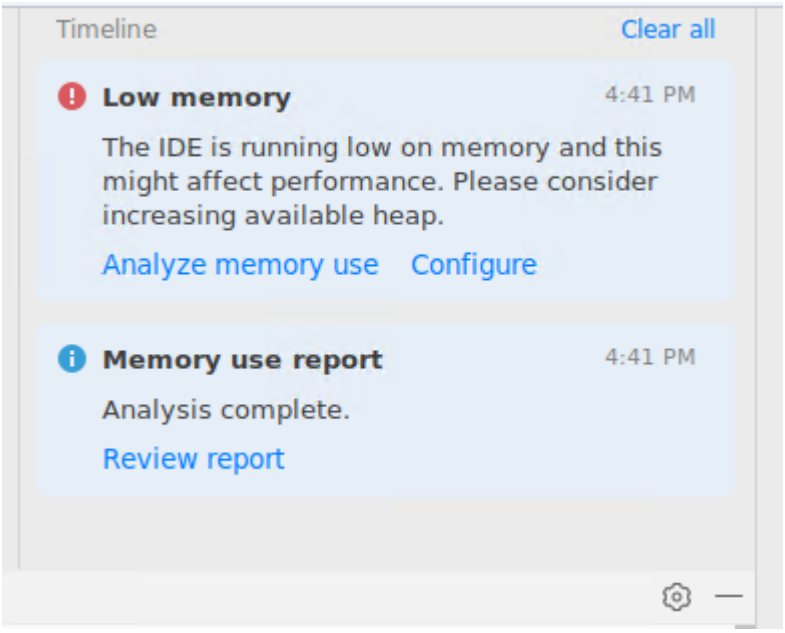
----结束

14.3.26 加载集群场景性能数据时提示“Low memory”

问题描述

MindStudio对采集后的集群场景性能原始数据进行解析并展示，数据加载时提示“Low memory”，如下图所示。

图 14-15 Low memory



问题分析

由于集群场景下采集到的原始性能数据较大，数据加载到内存中需要占用更多空间，导致MindStudio加载过程中出现异常提示。

解决方案

步骤1 单击顶部菜单栏“Help > Edit Custom VM Options...”。

步骤2 编辑虚拟机内存大小，调整到原虚拟机内存的5-10倍。如原内存为2048MB，可设置内存为20480MB，如图14-16所示。

图 14-16 设置虚拟机内存

步骤3 完成配置并保存，再重启MindStudio。

----结束

14.3.27 使用模型可视化功能时界面无法显示，且后台有报错信息提示

问题描述

在Ubuntu 20.04-aarch64环境上，使用Model Visualizer、Dump Configuration等模型可视化功能时界面无法显示，且后台提示如下两种报错信息任意一种。

1. SEVERE - #c.i.u.j.JBCefApp - /xxx/libjcef.so: /xxx/libcef.so: cannot allocate memory in static TLS block。
2. SEVERE - #c.i.o.p.Task - JCEF is not supported in this env or failed to initialize。

问题分析

1. libcef.so文件加载冲突。
2. libcef.so文件初始化失败。

解决方案

配置MindStudio环境变量。

步骤1 在MindStudio安装路径下的bin目录中执行**vi MindStudio.sh**命令，打开MindStudio.sh启动脚本文件，在文件第167行后面添加以下内容（以非root用户为例）。

```
export LD_PRELOAD=MindStudio安装目录/jbr/lib/libcef.so:$LD_PRELOAD
export LD_LIBRARY_PATH=MindStudio安装目录/jbr/lib/libcef.so:$LD_LIBRARY_PATH
```

步骤2 执行**:wq!**命令保存文件并退出。

步骤3 在MindStudio安装路径下的bin目录中使用如下命令重启MindStudio。

```
./MindStudio.sh
```

----结束

14.3.28 MindStudio 异常关闭后再次打开时，文件选择框不能加载文件以及界面按钮处于置灰状态

问题描述

在使用MindStudio时出现异常关闭后，再次打开MindStudio，文件选择框不能加载文件以及界面按钮处于置灰状态。

问题分析

MindStudio异常关闭后缓存文件受损。

解决方案

步骤1 重启MindStudio，是否可以正常使用。

- 是，问题解决。
- 否，请执行**步骤2**。

步骤2 配置环境变量。

- Linux系统配置方法如下：
  a. 以运行用户在任意目录下执行**vi ~/.bashrc**命令，打开**.bashrc**文件，在文件最后一行后面添加以下内容（以非root用户的默认安装路径为例）。
  ```
  export VFS_REBUILD=true
  ```
  b. 执行**:wq!**命令保存文件并退出。
  c. 执行**source ~/.bashrc**命令使其立即生效。
- Windows系统配置方法如下：
  a. 在Windows 10操作系统的“控制面板 > 系统和安全 > 系统”中选择“高级系统设置”。
  b. 弹出“系统属性”对话框，在“高级”页签中选择“环境变量(N)...”，如**图14-17**所示。

图 14-17 环境变量

c. 在“环境变量 > 系统变量(S)”弹框单击中下方的“新建(W)...”。

d. 弹出“新建系统变量”对话框，在“变量名(N)”中输入“VFS_REBUILD”，“变量值(V)”中输入“true”，单击“确认”。

e. 配置完成后，在“环境变量”对话框单击“确定”保存并关闭对话框。

----结束

14.3.29 MindStudio 不能正常使用 C&C++工程调试功能

问题描述

MindStudio打开C&C++工程进行调试时，发现调试时程序断点不能正常工作，出现不按照代码逻辑执行的现象。

问题分析

C&C++工程设置了编译优化选项-O2，该选项会在编译时对程序代码进行优化，编译后的程序与实际上的源码不匹配导致调试时无法正常断点，就会出现不按照代码逻辑执行的现象。

解决方案

调试时将cmakelist文件中的编译优化选项-O2改为-O0，或者直接删除。

