ICS 35.040
CCS L 71

团　　体　　标　　准

T/AI 131.1—2025

人工智能 算子接口
第 1 部分：基础数学类

Artificial intelligence — Operater interface—Part 1: Basic mathematical operators

2025 - 04 - 27 发布　　2025 - 04 - 27 实施

中关村视听产业技术创新联盟　　发 布

目　次

前　言

本文件按照 GB/T 1.1-2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

本文件为T/AI 131《人工智能 算子接口》的第1部分。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由新一代人工智能产业技术创新战略联盟AI标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：北京大学、北京大学长沙计算与数字经济研究院、中国电子技术标准化研究院、鹏城实验室、中国科学院软件研究所、北京百度网讯科技有限公司、华为技术有限公司、长沙壹零壹壹科技有限公司、中科寒武纪科技股份有限公司、上海商汤智能科技有限公司、浪潮电子信息产业股份有限公司、北京交通大学。

本文件主要起草人：杨超、勾海鹏、胡晓光、樊春、陈军、鲍薇、杨雨泽、敖玉龙、李克森、马艳军、于佃海、范睿博、贾梦珠、段炼、李敏、马银萍、付振新、黎子毅、龙汀汀、张云飞、关贺、胡帅、赵海英、张伟民、马珊珊、张军、李卉、郑若琳、栾晓旭、唐轶男、王丽、吴庚、钱晨、沈芷月、宋文林、刘文枫、高歌、聂简荻、陈德良、王新民、刘伟、杨征、冯海军、崔晓冉、汪群博。

引　言

近年来，我国人工智能行业发展呈现繁荣态势，相关软件和硬件均向多元化发展，云服务器、边缘设备、终端设备等不同类型的处理器层出不穷，与此同时，各类计算库、中间表示工具以及编程框架也呈现百花齐放的态势。AI软硬件的极大丰富，为应用的高效部署提供了诸多便利，但同时也带来了多样化、复杂化、碎片化的挑战。一方面，AI软件从业者需要考虑软件与各种不同主流人工智能处理器之间的适配，花费大量精力提升软件的可移植性，另一方面，每一款AI硬件的研发，都必须从底层对常用人工智能软件提供支持，否则难以融入现有的软件生态。这种M*N级别的软硬件映射关系，已经逐渐发展成桎梏人工智能应用发展的一大障碍。

人工智能算子是构建人工智能应用的基础运算，是相关硬件操作的封装，人工智能软件通过调用算子接口来使用硬件资源完成计算，算子接口是人工智能软硬件衔接的桥梁。制定T/AI 131《人工智能算子接口》，是对人工智能算子的核心数据结构、功能和接口参数的规范化和标准化，是降低人工智能软硬件适配难度、促进生态建设的基础性工作。

T/AI 131《人工智能 算子接口》拟由五个部分构成。

——第1部分：基础数学类。目的在于确立适用于人工智能算子接口的总则与核心数据结构，以及规范基础数学类算子接口的基本功能和参数的要求。

——第2部分：神经网络类。目的在于规范神经网络类算子的基本功能和参数的要求。

——第3部分：机器学习类。目的在于规范机器学习类算子的基本功能和参数的要求。

——第4部分：大模型类算子。目的在于规范大模型类算子的基本功能和参数的要求。

——第5部分：自动化测试框架。目的在于为算子接口提供自动化测试方法和参考实现，验证算子开发标准符合性。

人工智能　算子接口　第1部分：基础数学类

1　范围

本文件规定了面向人工智能领域的基础数学类算子接口的基本功能及参数要求。

本文件适用于人工智能基础数学类算子库的设计、开发与应用，以及相关软硬件及系统的研制。

2　规范性引用文件

本文件没有规范性引用文件。

3　术语和定义

下列术语和定义适用于本文件。

3.1

算子　operator

人工智能相关硬件操作的封装实现，是构建人工智能应用的基础组成部分。

3.2

算子接口　operator interface

是一套标准化的规定，用于描述和实现各类数学运算、逻辑运算或其他复杂计算单元的接口。

注：算子接口定义了一套规范化的API，使得开发人员能够以一致的方式来调用这些运算，而不必关心底层的具体实现细节。

3.3

封装　encapsulation

将数据和与数据相关的操作绑定在一起，形成一个独立单元的过程。

注：它对外隐藏其内部实现细节，可通过接口与外部进行交互。通过封装，可以增强代码的可读性、可维护性和可复用性。

3.4

张量 tensor

由同一类型元素所组成的多维数组。

3.5

稠密张量 dense tensor

元素全部或大部分为非零值的张量。

注：稠密张量一般采用非压缩的存储方式，即稠密存储，按照某种顺序存储张量的所有元素。

3.6

稀疏张量 sparse tensor

元素全部或大部分为零值的张量。

注：稀疏张量一般采用压缩的存储方式，即稀疏存储，只存储张量的非零值元素。

3.7

张量切片 tensor slice

指从张量中选取或提取部分元素的操作。假定张量x是形状为$[D_0, D_1, \ldots, D_{n-1}]$的n维张量，那么$x_{[d_0, d_1, \ldots, d_i, \ldots, d_k]}$是指取x在前k个维度上索引分别为$d_0, d_1, \ldots, d_i, \ldots, d_k$的元素集合，其中$0 \leq d_i < D_i$（对于所有$0 \leqslant i < k \leq n$且i为整数），它的返回结果为数值或形状为$[D_k, D_{k+1}, D_{k+2}, \ldots, D_{n-1}]$的$n-k$维张量。本文中张量切片也可写作为$x[d_0, d_1, \ldots, d_i, \ldots, d_k]$。

注：如果上述k等于n，那么返回结果为数值，该操作也被称为“张量索引(tensor index)”。

3.8

人工智能加速处理器 artificial intelligence accelerator processor

具备适配人工智能算法的运算微架构，能够完成人工智能应用运算处理的集成电路元件。

3.9

批 batch

训练样本的一部分。

3.10

广播 broadcasting

针对形状不同的两个张量进行逐元素运算时可采用的方法。

4 缩略语

下列缩略语适用于本文件。

ASIC:专用集成电路(Application Specific Integrated Circuit)

CPU:中央处理单元(Central Processing Unit)

FPGA:现场可编程门阵列(Field Programmable Gate Array)

GPU:图形处理单元(Graphic Processing Unit)

NPU:神经网络处理器(Neural-network Processing Unit)

5 总则

5.1 起始下标

本文件所涉及的数据或数据结构中的元素索引编号和计数应从0开始。

5.2 参数信息

5.2.1 本文件定义的算子接口参数列表、参数顺序、返回值描述方式等为算子接口的一种结构化定义，在实际实现过程中可根据应用情况进行调整。

5.2.2 本文件定义的算子接口参数列表为实现该算子接口的最小参数集合，在实际实现过程中可根据应用情况进行参数的扩充。

5.2.3 本文件对算子接口的参数定义了必选与可选两类，参数顺序执行必选参数在前，可选参数在后。其中必选参数表示必须实现且使用接口时必须传入，可选参数表示必须实现但使用接口时可以不传（即有默认值）。

注：需要说明的是，本文件定义的“数组长度”、“随机种子”、“数据格式”可选参数可根据实际编程语言和应用选择是否实现。若实际编程接口中参数顺序与本文件不一致，但是参数完整性保持一致，也符合本标准。

5.2.4 本文件针对算子接口参数中的可选参数，设置了默认值。默认值是在用户没有对算子的可选参数进行赋值时而被算子采用的数值，其数值大小是根据目前市场主流上层应用而选取的，例如PyTorch、PaddlePaddle、MindSpore、TensorFlow等上层应用。

5.3 编程语言

算子接口可用多种编程语言实现，C语言参考定义参见附录 A。

5.4 自动广播

两个形状不同的张量进行逐元素运算需要进行自动广播，扩充维度数值较小的张量以匹配维度数值较大的张量来完成运算。

5.5 状态处理

算子执行时应返回状态码，状态码分为成功和未成功（即错误）两类。对于算子执行过程中不可修复的错误，应直接退出；对于可修复错误，应采用返回错误状态码方式，由算子接口使用者决定如何处理错误。返回错误状态码时应优先返回相对具体的错误状态码，部分状态码应符合表1的规定。

表 1 状态码

状态码名称	类别	状态码
操作成功	成功	STATUS_SUCCESS
类型不匹配	未成功（错误）	STATUS_TYPE_MISMATCH
维度不匹配		STATUS_DIMENSIONS_MISMATCH
对象未初始化		STATUS_UNINITIALIZED_OBJECT
非法参数		STATUS_INVALID_ARGUMENT
内存不足		STATUS_ALLOC_FAILED

表 1　状态码（续）

状态码名称	类别	状态码
超出范围	未成功（错误）	STATUS_OUT_OF_RANGE
其它内部错误		STATUS_INTERNAL_ERROR

5.6　接口一致性

稀疏张量和稠密张量的构建应采用不同的接口，而基于两种张量的各类操作可以采用统一的接口或是不同的接口，如果采用不同的接口，接口应添加标识。

5.7　泛型标量类型

对于定义算子接口时无法确定数据类型的参数，可将其定义为泛型标量类(Scalar)。泛型标量类型表示参数可以根据运行时的实际数据传入不同类型的数值。在接口实现过程中，若部分参数定义为泛型标量类型，则认为该接口实现符合本文件定义。

6　数据结构

6.1　概要

本文件使用张量作为核心数据结构，承载数据，张量数据结构应包含元素类型、形状信息、布局信息、设备信息及其他扩展信息。

6.2　元素类型

用于描述元素的数据类型。包括无符号整数：8位，16位，32位，64位；有符号整数：8位、16位、32位、64位；浮点实数：16位、32位、64位；浮点复数：（32+32）位、（64+64）位；布尔类型，字符串类型等。

6.3　形状信息

用于描述张量维数和每一维的大小。如果张量维数为n，每一维对应的大小为$d_0, d_1, d_2, \ldots, d_{n-1}$，则张量的形状可以表示为$[d_0, d_1, d_2, \ldots, d_{n-1}]$，最左边的为第1维，最右边的为第$n$维。例如：标量维数为0，形状为[]；向量维数为1，形状为$[d_0]$；矩阵维数为2，形状为$[d_0, d_1]$。

6.4　布局信息

用于描述张量的存储格式以及张量各个维度的逻辑顺序。存储格式包括稠密存储和稀疏存储。逻辑顺序指张量遍历读取和存储具体数据时，各个维度的优先顺序。

当使用稠密存储时，对于形状为$[d_0, d_1, d_2, \ldots, d_{n-1}]$，逻辑顺序为 [0,1, … , n-2,n-1]的n维向量，首先取逻辑顺序中最左边第1位元素r_0，对应到当前要遍历的维度d_{r_0}，按照$(0,1,2, \ldots, d_{r_0} - 1)$的次序存储，其次取逻辑顺序中最左边第2位元素$r_1$，对应到当前要遍历的维度$d_{r_1}$，按照$(0,1,2, \ldots, d_{r_1} - 1)$的次序存储，以此类推。例如：张量形状为[2,3]，逻辑顺序为[0,1]，则先取逻辑顺序最左侧的第一个元素$r_0 = 0$，对应维度$d_{r_0} = d_0 = 2$，其次取逻辑顺序中的第二个元素$r_1 = 1$，对应维度d_{r_1}=$d_1 = 3$，则张量数据物理上第一个元素对应的坐标为[0,0]，第二个元素对应的坐标为[1,0]，第三个元素对应的坐标为[0,1]，以此类推。

当使用稠密存储时，默认的逻辑顺序应为[n-1, n-2, …, 0]。

当使用稀疏存储时，标准实施者应对所采用的具体格式进行详细说明。

6.5 设备信息

用于描述张量数据存储和运算的设备类型和设备编号。其中设备类型包括CPU、GPU、NPU、FPGA和ASIC等任何支持AI操作的设备或人工智能加速处理器。如果存在多个同类型的设备，可通过指定设备编号来区分。

6.6 其它扩展

用于提供自定义的扩展功能，比如内存管理等相关信息。

7 基础数学类算子接口

7.1 接口列表

基础数学类算子接口列表见表2。

表 2 基础数学类算子接口列表

类别	名称
张量创建与销毁	拷贝已有数据创建稠密张量，引用已有数据创建稠密张量，创建全零稠密张量，按指定值创建稠密张量，创建未初始化张量，创建连续内存张量，以均匀分布随机数创建稠密张量，以正态分布随机数创建稠密张量，以伯努利分布创建稠密张量，以多项式分布创建稠密张量，以数字序列创建稠密张量，创建线性空间均匀分布稠密张量，创建稀疏张量，创建量化张量，复制张量，复制对角线元素创建张量，销毁张量
张量查询与检查	形状查询，有限检查，无穷检查，未定义数检查，元素个数检查，秩查询，连续内存检查
张量转换	转换数据类型，改变张量形状，扩展维度，删除维度，张量转置，张量分拆，张量合并，张量堆叠，张量拆堆，张量切片，张量重复，张量补全，张量逆序变换，张量循环滚动变换，张量形状裁剪，张量数值裁剪，张量聚集，张量发散更新，扩展张量，展平张量，张量翻转，张量正负判断，条件判断组合张量，一维张量扩充，张量选择
算术操作	张量加法操作，张量减法操作，张量乘法操作，张量乘加操作，张量除法操作，张量整除操作，张量真除法操作，张量取模操作，张量逐元素取最大值，张量逐元素取最小值，张量绝对值操作，张量取倒数操作，张量对角线元素求和操作
比较操作	判断张量是否相等，判断张量是否不等，判断张量是否大于，判断张量是否大于等于，判断张量是否小于，判断张量是否小于等于，判断张量是否值相近

表 2　基础数学类算子接口列表（续）

类别	名称
逻辑操作	张量的“逻辑与”操作，张量的“逻辑或”操作，张量的“逻辑非”操作，张量的“逻辑异或”操作
位操作	逐位“与”操作，逐位“或”操作，逐位“异或”操作，逐位反转操作，逐位左移操作，逐位右移操作
幂操作	幂，平方根，平方根倒数，平方数
舍入操作	向下取整，向上取整，截断取整，就近取整，通用舍入取整
三角函数	正弦函数，余弦函数，正切函数，反正弦函数，反余弦函数，反正切函数
双曲函数	双曲正弦函数，双曲余弦函数，双曲正切函数，反双曲正弦函数，反双曲余弦函数，反双曲正切函数
指对函数	指数函数，指数函数扩展，以e为底的对数函数，以e为底的对数函数扩展，以10为底的对数函数，以2为底的对数函数
规约操作	规约，前缀和
索引操作	最大索引，最小索引，排序索引，Top K索引，非零索引
复数操作	复数构建、复数共轭、获取虚部、获取实部
信号处理	快速傅里叶变换、快速逆傅里叶变换
线性代数操作	矩阵乘法，向量内积， LU矩阵分解，Cholesky矩阵分解， QR矩阵分解， SVD奇异值分解，线性方程组求解， 最小二乘，矩阵求逆，求特征值以及特征向量，矩阵范数，线性操作，双线性操作
插值操作	插值操作

7.2　接口操作和参数

7.2.1　张量创建与销毁

7.2.1.1　拷贝已有数据创建稠密张量

7.2.1.1.1　功能

创建张量，若初始化数组不为空，则拷贝数组的值到张量空间进行初始化，否则创建一个空张量。

7.2.1.1.2　接口参数

拷贝已有数据创建稠密张量函数前向接口应符合表3，C语言示例见A.2.1.1。

表 3 拷贝已有数据创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数

表 3 拷贝已有数据创建稠密张量函数参数列表（续）

参数名	类型	可选/必选	描述
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，需保证初始化张量的设备类型和张量的设备类型一致，默认为CPU
张量形状	输入	必选	包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
初始化张量	输入	必选	如果为空，则表示不进行初始化操作
初始化数组的长度	输入	必选	以字节为单位，拷贝的数据量取决于张量空间和初始化数组之间尺寸较小者。若张量空间大于初始化数张量，则按初始化数组大小进行拷贝，张量中未被覆盖部分应被初始化为0。若张量空间小于初始化数组，则按照张量空间大小进行拷贝
输出张量	输出	必选	新创建的张量

7.2.1.1.3 接口返回值

没有错误：操作成功。
类型不匹配：张量的数据类型不一致。
非法参数：输入参数超出范围。
内存不足：创建张量分配空间不足。
维度不匹配：张量和初始化张量维度不匹配。
其它内部错误：内部调用操作出错。

7.2.1.2 引用已有数据创建稠密张量

7.2.1.2.1 功能

创建张量，对参数传入数组的数据进行引用。

7.2.1.2.2 接口参数

引用已有数据创建稠密张量函数前向接口应符合表4，C语言示例见A.2.1.2。

表 4 引用已有数据创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，需保证初始化张量的设备类型和张量的设备类型一致，默认为CPU
张量形状	输入	必选	包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
初始化张量	输入	可选	如果为空，则表示不进行初始化操作
输出张量	输出	必选	新创建的张量

7.2.1.2.3 接口返回值

没有错误：操作成功。

非法参数：权重与偏置张量的维度与输入输出张量不匹配。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.3 创建全零稠密张量

7.2.1.3.1 功能

创建元素值全为0的稠密张量。

7.2.1.3.2 接口参数

创建全零稠密张量函数前向接口应符合表5，C语言示例见A.2.1.3。

表 5 创建全零稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
张量形状	输入	必选	包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
输出张量	输出	必选	新创建的张量

7.2.1.3.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.4 按指定值创建稠密张量

7.2.1.4.1 功能

创建元素为某指定值的稠密张量。

7.2.1.4.2 接口参数

按指定值创建稠密张量函数前向接口应符合表6，C语言示例见A.2.1.4。

表 6 按指定值创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
张量形状	输入	可选	包括张量的维度，维度大小数组，以及布局信息， 其中布局信息没有指定时采用默认布局

表 6　按指定值创建稠密张量函数参数列表（续）

参数名	类型	可选/必选	描述
初始化值	输入	可选	用来初始化的数据，要求与参数“数据类型”所对应数据类型兼容
输出张量	输出	必选	新创建的张量

7.2.1.4.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.5　创建未初始化张量

7.2.1.5.1　功能

根据指定的类型、设备和形状创建张量。创建的张量中的数据为申请到的内存原本的数据内容，未经归零或者初始化，可能是任意合法的值。

7.2.1.5.2　接口参数

创建未初始化张量函数前向接口应符合表7，C语言示例见A.2.1.5。

表 7　创建未初始化张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	表示新张量的数据类型，可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
形状维度	输入	必选	新张量的形状维度
输出张量	输出	必选	新创建的张量

7.2.1.5.3　接口返回值

没有错误：操作成功。
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.1.6　创建连续内存张量

7.2.1.6.1　功能

如果输入张量的数据在内存中是连续存放的，则不做处理返回这个连续的输入张量，否则会将当前张量的数据复制到一段连续的内存上，返回复制后拥有连续内存的张量。

7.2.1.6.2　接口参数

生成连续内存张量函数前向接口应符合表8，C语言示例见A.2.1.6。

表 8　生成连续内存张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示具有连续内存的输出张量

7.2.1.6.3　接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.1.7　以均匀分布随机数创建稠密张量

7.2.1.7.1　功能

创建的稠密张量中的每个元素均是从符合对应均匀分布中随机生成的。

7.2.1.7.2　接口参数

以均匀分布随机数创建稠密张量函数前向接口应符合表9，C语言示例见A.2.1.7。

表 9　以均匀分布随机数创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
张量形状	输入	必选	包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
最小值	输入	可选	生成随机数所遵循均匀分布最小值，要求与参数“数据类型”所对应数据类型兼容，默认参数类型能表达的最小值
最大值	输入	可选	生成随机数所遵循均匀分布最大值，要求与参数“数据类型”所对应数据类型兼容，默认参数类型能表达的最大值
随机种子	输入	可选	生成随机数的种子。若随机种子为0，表示使用系统的随机种子；否则，使用随机种子来生成随机数，默认系统种子
输出张量	输出	必选	新创建的张量

7.2.1.7.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.8　以正态分布随机数创建稠密张量

7.2.1.8.1　功能

创建的稠密张量中的每个元素均是从符合对应正态分布中随机生成的。

7.2.1.8.2 接口参数

以正态分布随机数创建稠密张量函数前向接口应符合表10，C语言示例见A.2.1.8。

表 10 以正态分布随机数创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为浮点实数、浮点复数等，默认浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认CPU
张量形状	输入	必选	包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
均值	输入	可选	生成随机数所遵循正态分布均值，要求与参数“数据类型”所对应数据类型兼容，默认为0
标准差	输入	可选	生成随机数所遵循正态分布标准差，要求与参数“数据类型”所对应数据类型兼容，默认为1
随机种子	输入	可选	生成随机数的种子，若随机种子为0，表示使用系统的随机种子；否则，使用随机种子来生成随机数，默认系统种子
输出张量	输出	必选	新创建的张量

7.2.1.8.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.9 以伯努利分布创建稠密张量

7.2.1.9.1 功能

创建服从伯努利分布（0-1分布）的稠密张量。

7.2.1.9.2 接口参数

创建伯努利分布稠密张量函数前向接口应符合表11，C语言示例见A.2.1.9。

表 11 创建伯努利分布稠密张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入的概率值，为一个张量，类型为浮点数
输出张量	输出	必选	服从伯努利分布的随机张量，形状和数据类型与输入相同

7.2.1.9.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.10 以多项式分布创建稠密张量

7.2.1.10.1 功能

以输入张量伪概率，创建服从多项式分布的张量。

7.2.1.10.2 接口参数

以多项式分布创建稠密张量函数前向接口应符合表12，C语言示例见A.2.1.10。

表 12 以多项式分布创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入的概率值，为一个张量，类型为浮点数
采样次数	输入	可选	表示采样的次数，默认值为1
采样方式	输入	可选	表示是否为有放回采样，默认值为是
输出张量	输出	必选	服从多项式分布的随机张量，形状和数据类型与输入相同

7.2.1.10.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.11 创建随机排列的一维稠密张量

7.2.1.11.1 功能

从[0, 最大值)范围内取出每个整数，随机排列后组成一个一维稠密张量。

7.2.1.11.2 接口参数

创建随机排列（randperm)的一维张量函数前向接口应符合表13，C语言示例见A.2.1.11。

表 13 创建随机排列（randperm)的一维张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数和浮点复数，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
张量长度	输入	必选	表示张量的长度，也可表示范围的最大值，应该大于0
输出张量	输出	必选	新创建的张量

7.2.1.11.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.12 以数字序列创建稠密张量

7.2.1.12.1 功能

创建一维稠密张量的数字序列，张量中的值以初始值开始，按步长进行扩展到上限值为止。

7.2.1.12.2 接口参数

以数字序列创建稠密张量函数前向接口应符合表14，C语言示例见A.2.1.12。

表 14 以数字序列创建稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数和浮点复数，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
初始值	输入	必选	创建数字序列的第一个值，要求与参数“数据类型”所对应数据类型兼容
上限值	输入	必选	创建数字序列的上限，要求与参数“数据类型”所对应数据类型兼容
步长	输入	必选	生成的相邻两个随机数的变化步长，要求与参数“数据类型”所对应数据类型兼容
输出张量	输出	必选	新创建的张量

7.2.1.12.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.13 创建线性空间均匀分布稠密张量

7.2.1.13.1 功能

创建一维稠密张量，张量中的值是在区间起点和区间终点之间，均匀间隔的N个数。

7.2.1.13.2 接口参数

创建线性空间均匀分布稠密张量函数前向接口应符合表15，C语言示例见A.2.1.13。

表 15 创建线性空间均匀分布稠密张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为浮点实数和浮点复数，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
区间起点	输入	必选	区间的起点，包含在区间内，应与数据类型参数所对应数据类型兼容
区间终点	输入	必选	区间的终点，包含在区间内，应与数据类型参数所对应数据类型兼容
数据个数	输入	必选	创建张量的元素个数
输出张量	输出	必选	1维输出张量，长度为指定区间内数据的个数

7.2.1.13.3 接口返回值

没有错误：操作成功。
非法参数：由起点、终点、数据个数指定的区间不合法。

分配空间失败：创建张量分配空间不足。

7.2.1.14 创建稀疏张量

7.2.1.14.1 功能

创建稀疏张量，包括二维及高维稀疏张量的创建。

7.2.1.14.2 接口参数

创建稀疏张量函数前向接口应符合表16，C语言示例见A.2.1.14。

表 16 创建稀疏张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU
张量形状	输入	必选	张量的形状，包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
非零元坐标	输入	必选	创建稀疏张量中非零元的坐标，是二维（非零元个数，张量维度数）数组
非零元值数组	输入	必选	非零元素值组成的一维数组，长度为“非零元个数”
非零元个数	输入	必选	非零元的个数
输出张量	输出	必选	新创建的张量

7.2.1.14.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作

7.2.1.15 创建量化张量

7.2.1.15.1 功能

创建量化张量，若有初始化数组则用其值进行初始化，否则创建一个空张量。

7.2.1.15.2 接口参数

创建量化张量函数前向接口应符合表17，C语言示例见A.2.1.15。

表 17 创建量化张量函数参数列表

参数名	类型	可选/必选	描述
数据类型	输入	可选	可以为8位有符号对称量化整数和16位有符号对称量化整数、8位无符号非对称量化整数、8位有符号非对称量化整数、16位无符号非对称量化整数和16位有符号非对称量化整数、8位有符号通道级对称量化整数和16位有符号通道级对称量化整数，默认值为16位有符号对称量化整数
设备类型	输入	可选	可以为CPU、GPU、NPU、FPGA和ASIC等，默认为CPU

表 17　创建量化张量函数参数列表（续）

参数名	类型	可选/必选	描述
张量形状	输入	必选	张量的形状，包括张量的维度，维度大小数组，以及布局信息，其中布局信息没有指定时采用默认布局
初始化张量	输入	可选	初始化量化张量时，量化数据张量需要与参数“数据类型”兼容，如果为空，则表示不进行初始化操作
初始化值数组长度	输入	可选	初始化张量数组的长度，以字节为单位。拷贝的数据量取决于张量空间和初始化数组之间尺寸较小者。若张量空间大于初始化数组，则按初始化数组大小进行拷贝，张量中未被覆盖部分应被初始化为0。若张量空间小于初始化数组，则按照张量空间大小进行拷贝，默认为0
缩放因子	输入	必选	缩放因子为大于零的浮点类型数组，如果为张量级量化，则缩放因子的长度为1；如果通道级量化，则缩放因子的长度为张量形状中描述的通道数
零点	输入	可选	在水平方向上的移动量。如果数据类型为采用对称方式的量化数据类型，则零点被强制设置为0，真实浮点数x与量化值q的关系：x=(q-zero_point) * scale
输出张量	输出	必选	新创建带有量化信息缩放因子和零点的量化张量

7.2.1.15.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
维度不匹配：张量和初始化数组维度不匹配。
其它内部错误：内部调用操作出错。

7.2.1.15.4　其他附加说明

张量级（tensor-wise）量化：张量内部所有数据共享同一个scale。支持的数据类型有： 8位有符号对称量化整数和16位有符号对称量化整数，可选支持8位无符号非对称量化整数、 8位有符号非对称量化整数、 16位无符号非对称量化整数和16位有符号非对称量化整数。

通道级（channel-wise）量化：张量内部同一通道的数据共享同一个scale。支持的数据类型有：8位有符号通道级对称量化整数和16位有符号通道级对称量化整数。

非对称量化应符合式（1）-式子（3）：

$$r = \min(\max(x, a), b) \quad (1)$$

式中：
min-表示取两个数之间的最小值。
max-表示取两个数之间的最大值。
x-待被量化的浮点值；
a-待量化浮点数中的最小值；
b-待量化浮点数中的最大值；
r-表示被限定在区间[a,b]的待量化浮点数

$$s = \frac{b-a}{n-1} \quad (2)$$

式中：

b-待量化浮点数中的最大值；
a-待量化浮点数中的最小值；
n-表示具体量化级别k（例如8比特）所对应的最大值，即2^k；
S-表示缩放因子；

$$q = \left\lfloor \frac{r-a}{s} \right\rceil \quad (3)$$

式中：
r-表示被限定在区间[a,b]的待量化浮点数；
S-表示缩放因子；
⌊m⌉——表示将m向下取整到最近的整数。

对称量化应符合式（4）-式（5）：

$$M = max\,(abs(x)) \quad (4)$$

式中：
max--表示取两个数之间的最大值；
abs--表示对数值取绝对值。

$$q = \left\lfloor \frac{x}{M} * (n-1) \right\rceil \quad (5)$$

式中：
x——待被量化的浮点值；
M——待量化浮点数中的最大值（可取到）；
n-表示具体量化级别k（例如8比特）所对应的最大值，即2^k；
q——量化得到的整数；
⌊m⌉——表示将m四舍五入到最近的整数。

无论是对称量化还是非对称量化，应符合式（6）。

$$x = (q - zero\_point) * scale \quad (6)$$

式中：
x——待被量化的浮点值；
q——量化得到的整数；
$zero\_point$——零点值，对称量化即zero_point=0的特殊情况；
$scale$——缩放因子。

7.2.1.16 复制张量

7.2.1.16.1 功能

对张量进行复制操作。

7.2.1.16.2 接口参数

复制张量函数前向接口应符合表18，C语言示例见A.2.1.16。

表 18　复制张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	被复制的张量
输出张量	输出	必选	复制操作后的张量

7.2.1.16.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.16.4　其他附加说明

新张量的数据是被复制张量的拷贝，但是存放在新分配的空间中。

7.2.1.17　复制对角线元素创建张量

7.2.1.17.1　功能

创建一个对角矩阵，并使用指定1维张量来初始化对角线元素的值。

7.2.1.17.2　接口参数

复制对角线元素函数前向接口应符合表19，C语言示例见A.2.1.17。

表 19 复制对角线元素函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示维度输入张量，形状为[N]
输出张量	输出	必选	表示2维输出张量，形状为[N, N]

7.2.1.17.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。
内存不足：创建张量分配空间不足。
其它内部错误：内部调用操作出错。

7.2.1.18　销毁张量

7.2.1.18.1　功能

销毁某个指定张量。

7.2.1.18.2　接口参数

销毁张量函数前向接口应符合表20，C语言示例见A.2.1.18。

表 20　销毁张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	待销毁的张量

7.2.1.18.3　函数返回值

没有错误：操作成功。
非法参数：表示参数出错。
对象未初始化：表示传入的为空指针，无法销毁。
其它内部错误：内部调用操作出错。

7.2.2　张量查询与检查

7.2.2.1　形状查询

7.2.2.1.1　功能

获取张量的形状。

7.2.2.1.2　接口参数

形状查询函数前向接口应符合表21，C语言示例见A.2.2.1。

表 21 形状查询函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示输入张量形状的1维张量，整数类型

7.2.2.1.3　接口返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.2　有限检查

7.2.2.2.1　功能

检查张量数值是否是有限的，即不包含未定义数值或者无穷值（NaN或inf）。若不包含，则设置output[0]=true，否则设置output[0]=false。

7.2.2.2.2　接口参数

有限检查函数前向接口应符合表22，C语言示例见A.2.2.2。

表 22 有限检查函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示检查的结果张量，形状与输入张量形状一致，元素为布尔类型

7.2.2.2.3　函数返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.3 无穷检查

7.2.2.3.1 功能

检查张量是否包含无穷值inf，有则返回true。

7.2.2.3.2 接口参数

无穷检查函数前向接口应符合表23，C语言示例见A.2.2.3。

表 23 无穷检查函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示检查的结果张量，形状与输入张量形状一致，元素为布尔类型

7.2.2.3.3 函数返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.4 未定义数检查

7.2.2.4.1 功能

检查张量是否包含未定义数值NaN，有则返回true。

7.2.2.4.2 接口参数

未定义数检查函数前向接口应符合表24，C语言示例见A.2.2.4。

表 24 未定义数检查函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示检查的结果张量，形状与输入张量形状一致，元素为布尔类型

7.2.2.4.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.5 元素个数查询

7.2.2.5.1 功能

获取张量元素的个数。

7.2.2.5.2 接口参数

元素个数查询函数前向接口应符合表25，C语言示例见A.2.2.5。

表 25 元素个数查询函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出实数	输出	必选	表示输入张量的元素个数

7.2.2.5.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.6 秩查询

7.2.2.6.1 功能

获取张量的秩。

7.2.2.6.2 接口参数

秩查询函数前向接口应符合表26，C语言示例见A.2.2.6。

表 26 秩查询函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出实数	输出	必选	表示输入张量的秩

7.2.2.6.3 接口返回值

没有错误：操作成功。
非法参数：表示参数出错。

7.2.2.7 连续内存检查

7.2.2.7.1 功能

判断输入张量的数据在内存中是否是连续存放的，若是，则返回布尔值 true，否则返回布尔值 false。

7.2.2.7.2 接口参数

连续内存检查函数前向接口应符合表27，C语言示例见A.2.2.7。

表 27 连续内存检查函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出布尔值	输出	必选	表示张量是否为连续

7.2.2.7.3 函数返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
非法参数：表示其他参数不合法。

其它内部错误：内部调用操作出错。

7.2.3 张量转换

7.2.3.1 转换数据类型

7.2.3.1.1 功能

转换张量的数据类型。输出张量的每个元素是将对应输入张量的元素转换为参数“数据类型”后得到的结果。输入张量原始类型和欲转换成的类型必须能够兼容。从复数转换成实数，只保留实部；从实数转换成复数，虚部为零。

7.2.3.1.2 接口参数

转换数据类型函数前向接口应符合表28，C语言示例见A.2.3.1。

表 28 转换数据类型函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
数据类型	输入	可选	表示要转换的类型，可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等，默认为浮点实数
输出张量	输出	必选	表示类型转换后的张量

7.2.3.1.3 接口返回值

没有错误：操作成功。

类型不匹配：表示输入张量对象的类型和要转换的类型不兼容。

对象未初始化：表示输入张量对象不合法

内存不足：表示输出向量分配空间不足

非法参数：表示其他参数不合法。

其它内部错误：内部调用操作出错。

7.2.3.2 改变张量形状

7.2.3.2.1 功能

改变张量的形状，可以通过in-place或out-of-place执行，调整后的张量元素数量必须与原张量的元素数量一致。

In-place执行条件：能够为新的维度找到一种布局，使得这种布局与张量原来的布局相兼容，在此布局下不需要改变底层数据物理存储的排布，可以直接修改维度信息，并赋予相应的兼容布局。

拷贝（out-of-place）：若不满足in-place执行的条件，则需要基于新的形状创建新的张量，进行数据拷贝，布局指定为默认布局。其中输入张量的读取和输出张量的写入按照各自布局规定的顺序进行。

布局兼容：为方便描述，我们把维度顺序称为布局，把遵循行优先的布局称为连续的布局。对于一个n维张量来说，若其布局为{n-1, n-2, … , 0}，则该布局是连续的。基于此，在张量的布局数组中，从左到右遍历，按1递减的部分就可以认为是连续的子布局。以一个5维张量T_0为例，假设其维度为$\{d_0, d_1, d_2, d_3, d_4\}$，布局数组为{3, 2, 4, 1, 0}(物理内存地址每加1时，按照d_3, d_2, d_4, d_1, d_0的顺序对

维度遍历)，则此布局最少可以分割为3个连续的子布局，即{{3, 2}, {4}, {1, 0}}，相当于其维度划分为三个子块{$\{d_0,d_1\},\{d_2,d_3\},\{d_4\}$}，对应的元素数量为{$\{d_0*d_1\},\{d_2*d_3\},\{d_4\}$}。因此从整体上看，张量$T_0$可以看做一个新的维度数量为3，维度大小为{$D_0,D_1,D_2$}（其中$D_0=d_0*d_1$，$D_1=d_2*d_3$，$D_2=d_4$），布局数组为{1, 2, 0}的张量。此时，**新的维度、布局**与**原维度、布局**是兼容的，两者相互转换时，可以进行inplace的reshape操作。在上述例子的基础上，若新的维度数量为4，维度大小为{D_0,D_1,D_2,D_3}（其中$D_0=d_0*d_1$，$D_1=d_2*d_3$，$D_2*D_3=d_4$），布局为{1, 3, 2, 0}，此时也是布局兼容的。

7.2.3.2.2 接口参数

改变张量形状函数前向接口应符合表29，C语言示例见A.2.3.2。

表 29 改变张量形状函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
维度数组	输入	必选	表示新形状每个维度对应的大小
维度数	输入	必选	表示维度数组的长度
输出张量	输出	必选	表示改变形状后的张量

7.2.3.2.3 接口返回值

没有错误：操作成功。
维度不匹配：表示输入张量对象的维度总大小和输出张量维度总大小不一致。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.3 扩展维度

7.2.3.3.1 功能

在输入张量的指定维度插入大小为1的新维度，但不改变其他维度和总的形状大小。

7.2.3.3.2 接口参数

扩展维度函数前向接口应符合表30，C语言示例见A.2.3.3。

表 30 扩展维度函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
插入位置	输入	可选	表示插入新维度的位置，默认为0
输出张量	输出	必选	表示插入新维度后的输出张量

7.2.3.3.3 接口返回值

没有错误：操作成功。
超出范围：表示插入位置超出输入张量的维度。

对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.4 删除维度

7.2.3.4.1 功能

在输入张量的指定维度删除为1的维度，但不改变其他维度和总形状大小。如果删除位置的维度不为1，不进行任何处理。

7.2.3.4.2 接口参数

删除维度函数前向接口应符合表31，C语言示例见A.2.3.4。

表 31 删除维度函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
删除位置数组	输入	可选	表示要删除维度的位置数组，默认为空
删除位置数	输入	可选	表示删除位置数组长度。如果是0，则处理所有维度，默认为0
输出张量	输出	必选	表示删除维度为1的输出张量

7.2.3.4.3 函数返回值

没有错误：操作成功。
超出范围：表示删除位置超出输入张量的维度。
对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.5 张量转置

7.2.3.5.1 功能

转置张量。如果置换维度数组不为空，则输入张量第i维会变成输出张量的第（置换维度数组[i]）维；如果置换维度数组为空，则当输入张量形状为(i[0], i[1], ... i[n-2], i[n-1])，则输出张量形状为(i[n-1], i[n-2], ... i[1], i[0])。

7.2.3.5.2 接口参数

张量转置函数前向接口应符合表32，C语言示例见A.2.3.5。

表 32 张量转置函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
置换维度数组	输入	可选	表示置换维度数组，默认为空

表 32 张量转置函数参数列表（续）

参数名	类型	可选/必选	描述
置换维度数组长度	输入	可选	表示置换维度数组长度，默认为0
输出张量	输出	必选	表示转换后的张量

7.2.3.5.3 函数返回值

没有错误：操作成功。
类型不匹配：表示输入张量对象的类型和要转换的类型不兼容
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.6 张量分拆

7.2.3.6.1 功能

在指定维度，将输入张量进行分拆。

7.2.3.6.2 接口参数

张量分拆函数前向接口应符合表33，C语言示例见A.2.3.6。

表 33 张量分拆函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
分拆大小数组	输入	必选	表示分拆大小数组。以split_sizes表示分拆大小数组，如果不为空，则split_sizes[i]>0表示第i个分拆大小，并且split_sizes元素总和必须等于输入张量形状总大小；如果为空，表示每个分拆大小相等，分拆数量由参数“分拆数量”确定
分拆数量	输入	必选	表示分拆数量，即和分拆大小数组长度相等
分拆维度	输入	可选	表示要分拆的维度，从0开始计数，默认为0
输出张量数组	输出	必选	表示分拆后所输出张量数组，其长度为参数“分拆数量”

7.2.3.6.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
超出范围：表示axis超出输入张量维度。
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.7 张量合并

7.2.3.7.1 功能

在指定维度，将输入张量数组进行合并。所有输入张量除了“合并维度”的维度可以不一样，其他维度的大小必须相等。如果“输入张量数组[i]”形状为$[\mathrm{D}_0,\mathrm{D}_1,\ldots,D^i_{axis},\ldots,\mathrm{D}_{\mathrm{ndim}-1}]$，那么合并后的形状$[\mathrm{D}_0,\mathrm{D}_1,\ldots,\sum D^i_{axis},\ldots,\mathrm{D}_{\mathrm{ndim}-1}]$。

7.2.3.7.2 接口参数

张量合并函数前向接口应符合表34，C语言示例见A.2.3.7。

表 34 张量合并函数参数列表

参数名	类型	可选/必选	描述
输入张量数组	输入	必选	表示输入的张量数组
输入张量数组长度	输入	必选	表示输入张量数组长度
合并维度	输入	必选	表示要合并的维度
输出张量	输出	必选	表示合并后的张量

7.2.3.7.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
超出范围：表示堆叠维度超出输入张量维度
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.8 张量堆叠

7.2.3.8.1 功能

在指定维度，将输入张量数组进行堆叠，输入张量形状都必须相同。如果输入张量数组每个元素维度为d，那么输出张量的维度为d+1，新增维度大小为“输入张量数组长度”。

7.2.3.8.2 接口参数

张量堆叠函数前向接口应符合表35，C语言示例见A.2.3.8。

表 35 张量堆叠函数参数列表

参数名	类型	可选/必选	描述
输入张量数组	输入	必选	表示需堆叠的输入张量数组
输入张量数组长度	输入	必选	表示输入张量数组长度
堆叠维度	输入	必选	表示要堆叠的维度
输出张量	输出	必选	表示输出张量

7.2.3.8.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法

超出范围：表示堆叠维度超出输入张量维度
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.9 张量拆堆

7.2.3.9.1 功能

在指定维度，将输入张量拆成“输出张量数组长度”个输出张量。如果输入张量维度为d，那么输出张量的维度为d-1。

7.2.3.9.2 接口参数

张量拆堆函数前向接口应符合表36，C语言示例见A.2.3.9。

表 36 张量拆堆函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示需拆堆的输入张量
拆堆维度	输入	可选	表示指定拆堆的维度，从0开始计数，默认为0
输出张量数组长度	输入	必选	表示输出张量数组的长度，值必须和输入张量在拆堆维度上的值相同
输出张量数组	输出	必选	表示拆叠后的输出张量数组，其长度为“输出张量数组长度”

7.2.3.9.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
超出范围：表示拆堆维度超出输入张量维度。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.10 张量切片

7.2.3.10.1 功能

从输入张量中提取片段，假设输入张量形状为$[D_0, D_1, \ldots, D_{ndim-1}]$，那么对于第i维来说，必须满足$0 \le begin[i] \le begin[i] + size[i] \le D_i$，其中begin为起始位置数组，size为切片大小数组。

7.2.3.10.2 前向接口参数

张量切片函数前向接口应符合表37，C语言示例见A.2.3.10。

表 37 张量切片函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
起始位置数组	输入	必选	表示每个维度提取起始位置数组，长度为输入张量的维度
切片大小数组	输入	必选	表示每个维度提取大小数组，长度为输入张量的维度

表 37　张量切片函数参数列表（续）

参数名	类型	可选/必选	描述
切片步长数组	输入	必选	表示每个维度上提取步长数组，长度为输入张量的维度
输出张量	输出	必选	表示输出张量

7.2.3.10.3　前向接口返回值

没有错误：操作成功。

对象未初始化：表示输入张量对象不合法。

超出范围：表示起始位置数组中元素或者切片大小数组中元素或者切片步长数组中元素超出输入张量维度大小。

内存不足：表示输出向量分配空间不足。

非法参数：表示其他参数不合法。

其它内部错误：内部调用操作出错。

7.2.3.10.4　后向接口参数

张量切片函数后向接口应符合表38，C语言示例见A.2.3.10。

表 38　张量切片函数后向接口参数列表

参数名	类型	可选/必选	描述
输出梯度	输入	必选	表示输出张量的梯度张量
输入张量	输入	可选	表示输入张量
起始位置数组	输入	可选	表示每个维度提取起始位置数组，长度为输入张量的维度
切片大小数组	输入	可选	表示每个维度提取大小数组，长度为输入张量的维度
切片步长数组	输入	可选	表示每个维度上提取步长数组，长度为输入张量的维度
输入梯度	输出	必选	表示输入张量的梯度张量

7.2.3.10.5　后向接口返回值

没有错误：操作成功。

对象未初始化：表示输入张量对象不合法。

超出范围：表示起始位置数组中元素或者切片大小数组中元素或者切片步长数组中元素超出输入张量维度大小。

内存不足：表示输出向量分配空间不足。

非法参数：表示其他参数不合法。

其它内部错误：内部调用操作出错。

7.2.3.11　张量重复

7.2.3.11.1　功能

通过重复输入向量来构造输出向量。

7.2.3.11.2　接口参数

张量重复函数前向接口应符合表39，C语言示例见A.2.3.11。

表 39　张量重复函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
重复数组	输入	必选	表示每个维度重复次数，长度等于输入张量维度
输出张量	输出	必选	表示输出张量

7.2.3.11.3　函数返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.12　张量补全

7.2.3.12.1　功能

按照指定的模式来补全张量。

7.2.3.12.2　接口参数

张量补全函数前向接口应符合表40，C语言示例见A.2.3.12。

表 40　张量补全函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
补全宽度数组	输入	必选	如果输入张量维度为n，那么补全宽度数组长度为2*n。假设pad_width代表补全宽度数组，对于维度i来说，pad_width[2*i]和pad_width[2*i + 1]分别表示在第i维度数据之前补全的宽度和之后补全的宽度
补全数值	输入	可选	表示补全的数值，默认为0
补全模式	输入	可选	表示补全模式，枚举类型。如果是PAD_CONST，就采用“补全数值”来补全；如果是PAD_PERIOD模式，就采用张量本身数据周期补全；如果是PAD_MIRROR模式，就采用张量本身数据镜像补全，默认是PAD_CONST模式
输出张量	输出	必选	表示补全后的输出张量

7.2.3.12.3　接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.13　张量逆序变换

7.2.3.13.1 功能

在逆序位置数组指定的轴上，对输入张量进行数据的逆序操作。

7.2.3.13.2 接口参数

张量逆序变换函数前向接口应符合表41，C语言示例见A.2.3.13。

表 41 张量逆序变换函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
逆序位置数组	输入	必选	指定逆序运算的轴的数组，从0开始计数
逆序位置数组长度	输入	必选	表示逆序位置数组的长度
输出张量	输出	必选	表示输出张量，形状、数据类型与输入张量相同

7.2.3.13.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
超出范围：表示逆序位置数组超出输入张量维度
非法参数：表示其他参数不合法。

7.2.3.14 张量循环滚动变换

7.2.3.14.1 功能

沿指定的轴，对输入张量进行循环滚动变换。当元素移动到最后位置时，会插入到该维度上的第一个位置。

7.2.3.14.2 接口参数

张量循环滚动变换函数前向接口应符合表42，C语言示例见A.2.3.14。

表 42 张量循环滚动变换函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
滚动位移数组	输入	必选	表示shifts参数的长度。
滚动位移	输入	必选	每个维度上的滚动位移
滚动的轴	输入	可选	滚动的轴，可以指定多个维度。可以为空，此时输入张量会按照1维张量执行滚动变换。若不为空，则长度必须与滚动位移长度相同，默认为空
输出张量	输出	必选	输出张量，张量形状和数据类型与输入张量相同

7.2.3.14.3 接口返回值

没有错误：操作成功。
非法参数：设置的轴超出了输入张量的维数。

7.2.3.15 张量形状裁剪

7.2.3.15.1 功能

根据指定形状和偏移量，裁剪输入张量。

7.2.3.15.2 接口参数

张量形状裁剪函数前向接口应符合表43，C语言示例见A.2.3.15。

表 43 张量形状裁剪函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量形状数组	输入	必选	表示输出张量的形状，数组长度与输入张量的维数相同
偏移量数组	输入	可选	表示每个维度长裁剪的偏移量，数组长度与输入张量的维数相同，默认为0
输出张量	输出	必选	表示输出张量

7.2.3.15.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
非法参数：表示其他参数不合法。

7.2.3.16 张量数值裁剪

7.2.3.16.1 功能

对输入张量的数值进行裁剪，输出张量的数值被限制在区间[low,high)范围内，其中low为区间下限，high为区间上限，见式（7）。

$$output = \min(\max(input, low), high) \quad (7)$$

式中：
min——取最小值；
max——取最大值；
input——表示输入张量；
low——区间下限；
high——区间上限；
output——输出张量

7.2.3.16.2 前向接口参数

张量数值裁剪函数前向接口应符合表44，C语言示例见A.2.3.16。

表 44 张量数值裁剪函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
区间下限	输入	可选	表示区间的下限，输入张量中小于该值的元素将由该值代替，数据类型与输入张量在计算上兼容，默认为None，表示无下限
区间上限	输入	可选	表示区间的上限，输入张量中大于该值的元素将由该值代替，数据类型与输入张量在计算上兼容,默认为None,表示无上限
输出张量	输出	必选	表示输出张量，形状、数据类型与输入张量相同

7.2.3.16.3 前向接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
非法参数：表示其他参数不合法。

7.2.3.16.4 后向接口参数

张量数值裁剪函数后向接口应符合表45，C语言示例见A.2.3.16。

表 45 张量数值裁剪后向接口函数参数列表

参数名	类型	可选/必选	描述
输出梯度	输入	必选	表示输出张量的梯度张量
区间下限	输入	必选	表示区间的下限，输入张量中小于该值的元素将由该值代替，数据类型与输入张量在计算上兼容
区间上限	输入	必选	表示区间的上限，输入张量中大于该值的元素将由该值代替，数据类型与输入张量在计算上兼容
输入梯度	输出	必选	表示输入张量的梯度张量

7.2.3.16.5 后向接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
非法参数：表示其他参数不合法。

7.2.3.17 张量聚集

7.2.3.17.1 功能

根据索引张量获取输入张量对应维度的条目，并将它们拼接在一起。当索引张量是一个1-D维张量且元素个数为k时，输入张量是多维张量，第一个维度大小为m，可以表示为$[X_0, X_1, \ldots, X_m]$，那么输出张量是$[X_{index[0]}, X_{index[1]}, \ldots, X_{index[k]}]$，其中index是索引张量。当索引张量是一个K维张量，它可以认为是从输入张量中取K-1维张量，每一个元素是一个切片，见式（8）。

$$output[i_0, \ldots, i_{K-2}] = input[index[i_0, \ldots, i_{K-2}]] \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

式中：
input——输入张量；
index——索引张量；
output——输出张量

7.2.3.17.2 接口参数

张量聚集函数前向接口应符合表46，C语言示例见A.2.3.17。

表 46 张量聚集函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
索引张量	输入	必选	表示索引张量，维度必须大于1且小于等于输入张量的维度，可以为整数类型
输出张量	输出	必选	表示输出张量，数据类型与输入张量相同

7.2.3.17.3 函数返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
非法参数：表示其他参数不合法。

7.2.3.18 张量发散更新

7.2.3.18.1 功能

依据索引张量中的索引，将更新数据张量中的数据更新到输入张量中。当输入张量的形状为$[N_0, N_1, \ldots, N_k]$，索引张量是1维张量且元素个数为M时，更新数据张量的形状为$[M, N_1, \ldots, N_k]$。

7.2.3.18.2 接口参数

张量发散更新函数前向接口应符合表47，C语言示例见A.2.3.18。

表 47 张量发散更新函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
索引张量	输入	必选	表示索引张量，维度必须大于1且小于等于输入张量的维度，可以为整数类型
更新数据张量	输入	必选	表示更新数据张量
是否覆盖	输入	可选	如果索引张量中的索引值有重复，“是否覆盖”=true时旧更新值将被新更新值覆盖,“是否覆盖”=false时新更新值将与旧更新值相加,默认为true.
输出张量	输出	必选	表示输出张量，形状、数据类型与输入张量相同

7.2.3.18.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
非法参数：表示其他参数不合法。

7.2.3.19 扩展张量

7.2.3.19.1 功能

根据指定的形状扩展张量，扩展后，张量的形状和指定的形状保持一致。

7.2.3.19.2 接口参数

扩展张量函数前向接口应符合表48，C语言示例见A.2.3.19。

表 48 扩展张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
扩展后形状	输入	必选	表示新形状的大小
输出张量	输出	必选	表示形状扩展后的张量

7.2.3.19.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.20 展平张量

7.2.3.20.1 功能

根据给定的起始维度和结束维度将张量中的连续维度展平。

7.2.3.20.2 接口参数

展平张量函数前向接口应符合表49，C语言示例见A.2.3.20。

表 49 展平张量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
起始维度	输入	必选	表示需要展平的起始维度
结束维度	输入	必选	表示需要展平的结束维度
输出张量	输出	必选	表示展平后的张量

7.2.3.20.3 接口返回值

没有错误：操作成功。
维度错误：表示起始维度或者结束维度不合法。
对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.21 张量翻转

7.2.3.21.1 功能

沿指定轴翻转张量。

7.2.3.21.2 接口参数

张量翻转函数前向接口应符合表50，C语言示例见A.2.3.21。

表 50 张量翻转函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
翻转轴	输入	必选	表示需要翻转的轴
输出张量	输出	必选	表示翻转后的张量

7.2.3.21.3 接口返回值

没有错误：操作成功。
轴错误：表示轴不合法。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.22 张量正负判断

7.2.3.22.1 功能

对张量中的每个元素进行正负判断，正数转为1，负数转为-1，0保持不变。

7.2.3.22.2 接口参数

张量正负判断函数前向接口应符合表51，C语言示例见A.2.3.22。

表 51 张量正负判断函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
输出张量	输出	必选	表示正负判断后的张量

7.2.3.22.3 接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.23 条件判断组合张量

7.2.3.23.1 功能

根据bool型条件，选择x或y的元素组成多维张量，见式（9）。

$$z_i = \begin{cases} x_i \ , if\ cond_i\ is\ True \\ y_i \ , \text{if}\ cond_i\ is\ False \end{cases} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (9)$$

式中：
x——第一个输入张量；
y——第二个输入张量；
z——输出张量；
cond——表示bool条件一维张量数组。

7.2.3.23.2 接口参数

条件判断组合张量函数前向接口应符合表52，C语言示例见A.2.3.23。

表 52　条件判断组合张量函数参数列表

参数名	类型	可选/必选	描述
输入条件	输入	必选	表示选择x或y元素的条件
输入张量X	输入	必选	表示输入张量x
输入张量Y	输入	必选	表示输入张量y
输出张量	输出	必选	表示条件判断后的张量

7.2.3.23.3　接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.24　一维张量扩充

7.2.3.24.1　功能

输入是张量或者包含张量的列表，包含 k 个一维张量，形状分别为(N1,)，(N2,)，...，(Nk,)，对每个张量做扩充操作，输出 k 个 k 维张量，每个张量的形状均为(N1, N2, ...，Nk)。

7.2.3.24.2　接口参数

一维张量扩充函数前向接口应符合表53，C语言示例见A.2.3.24。

表 53　一维张量扩充函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示k个一维张量，形状分别为(N1,)，(N2,)，…，(Nk,)
输出张量	输出	必选	表示扩充后的k个形状为(N1，N2，…，Nk)的张量

7.2.3.24.3　接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.3.25 张量选择

7.2.3.24.4　功能

根据掩码张量选择出输入张量中的值，组成1维张量。

7.2.3.24.5　接口参数

张量选择函数前向接口应符合表54，C语言示例见A.2.3.24。

表 54　张量选择函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量
掩码张量	输入	必选	表示输入张量中需要选择的元素，为布尔类型
输出张量	输出	必选	表示根据掩码张量选择出输入张量中的值所组成的1维张量，数据类型与输入张量相同

7.2.3.24.6　接口返回值

没有错误：操作成功。
对象未初始化：表示输入张量对象不合法。
非法参数：表示其他参数不合法。

7.2.4　算术操作

7.2.4.1　张量加法操作

7.2.4.1.1　功能

两个张量逐元素求和，见式（10）。

$$z_i = x_i + y_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (10)$$

式中：
x——第一个输入张量；
y——第二个输入张量；
z——输出张量。

7.2.4.1.2　接口参数

张量加法操作函数前向接口应符合表55，C语言示例见A.2.4.1。

表 55　张量加法操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.1.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.4.2　张量减法操作

7.2.4.2.1　功能

两个张量逐元素求和，见式（11）。

$$z_i = x_i - y_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (11)$$

式中：
x——第一个输入张量；

y——第二个输入张量；
z——输出张量。

7.2.4.2.2 接口参数

张量减法操作函数前向接口应符合表56，C语言示例见A.2.4.2。

表 56 张量减法操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.2.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.4.3 张量乘法操作

7.2.4.3.1 功能描述

张量与张量间乘法操作。输出张量的每一个元素为输入张量的元素乘第二个输入张量对应的元素，即$z_i = x_i * y_i$。

7.2.4.3.2 接口参数

张量乘法操作函数前向接口应符合表57，C语言示例见A.2.4.3。

表 57 张量乘法操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.3.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.4.4 张量乘加操作

7.2.4.4.1 功能描述：

张量与张量间乘加操作。输出张量的每一个元素为输入张量的元素乘以第二个输入张量对应的元素，然后加上第三个输入张量的元素，见式（12）。

$$z_i = x_i * y_i + a_i \quad \text{（12）}$$

式中：
x——第一个输入张量；

y——第二个输入张量；

a——第二个输入张量；

z——输出张量。

7.2.4.4.2 接口参数

张量乘加操作函数前向接口应符合表58，C语言示例见A.2.4.4。

表 58 张量乘加操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.4.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.4.5 张量除法操作

7.2.4.5.1 功能

张量与张量间除法操作。输出张量的每一个元素为第一个输入张量的元素除第二个输入向量对应的元素，见式（13）。

$$z_i = x_i / y_i \quad \cdots\cdots (13)$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.4.5.2 接口参数

张量除法函数前向接口应符合表59，C语言示例见A.2.4.5。

表 59 张量除法函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.5.3 函数返回值

没有错误：操作成功。

非法参数：表示输入参数不合法。

类型不匹配：表示参数的数据类型不一致。

7.2.4.6 张量整除操作

7.2.4.6.1 功能

张量与张量间整除操作。输出张量的每一个元素为第一个输入张量的元素除第二个输入向量对应的元素，计算结果按照floor方式取整，见式（14）。

$$z_i = \lfloor x_i / y_i \rfloor \quad \text{（14）}$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.4.6.2 接口参数

张量整除函数前向接口应符合表60，C语言示例见A.2.4.6。

表 60 张量整除函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.6.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.4.7 张量真除法操作

7.2.4.7.1 功能

使用浮点数计算的张量除法操作，与参与操作的张量类型无关。

7.2.4.7.2 接口参数

张量真除法操作函数前向接口应符合表61，C语言示例见A.2.4.7。

表 61 张量真除法操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.7.3 接口返回值

没有错误：操作成功。

非法参数：张量形状不匹配或不满足广播要求。

7.2.4.8 张量取模操作

7.2.4.8.1 功能

计算两个输入张量逐元素相除得到的余数。

7.2.4.8.2 接口参数

张量取模操作函数前向接口应符合表62，C语言示例见A.2.4.8。

表 62 张量取模操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.8.3 接口返回值

没有错误：操作成功。
非法参数：表示输入参数不合法。
类型不匹配：表示参数的数据类型不一致。

7.2.4.9 张量逐元素取最大值

7.2.4.9.1 功能

张量与张量间逐元素比较，取最大值操作，见式（15）。

$$z_i = \max(x_i, y_i) \quad \text{（15）}$$

式中：
x——第一个输入张量；
y——第二个输入张量；
max——取最大值；
z——输出张量。

7.2.4.9.2 接口参数

张量逐元素取最大值函数前向接口应符合表63，C语言示例见A.2.4.9。

表 63 张量逐元素取最大值函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.9.3 函数返回值

没有错误：操作成功。
非法参数：表示输入参数不合法。
类型不匹配：表示参数的数据类型不一致。

7.2.4.10 张量逐元素取最小值

7.2.4.10.1 功能

张量与张量间逐元素比较，取最小值操作，见式（16）。

$$z_i = \min(x_i, y_i) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (16)$$

式中：

x——第一个输入张量；

y——第二个输入张量；

min——取最小值；

z——输出张量。

7.2.4.10.2 接口参数

张量逐元素取最小值函数前向接口应符合表64，C语言示例见A.2.4.10。

表 64 张量逐元素取最小值函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果

7.2.4.10.3 接口返回值

没有错误：操作成功。

非法参数：表示输入参数不合法。

类型不匹配：表示参数的数据类型不一致。

7.2.4.11 张量绝对值操作

7.2.4.11.1 功能

计算输入张量的绝对值（逐元素求绝对值）。其中，输出也是一个张量，张量的每一个元素由输入张量对应元素的绝对值得到。当输入张量中的每个元素的类型为基本数据类型时，$y_i = |x_i|$， 当张量中的每个元素为复数类型（$x_i = a + bj$）时，$y_i = \sqrt{a^2 + b^2}$。

7.2.4.11.2 接口参数

张量绝对值操作函数前向接口应符合表65，C语言示例见A.2.4.11。

表 65 张量绝对值操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输出张量	输出	必选	表示计算结果

7.2.4.11.3 接口返回值

没有错误：操作成功。

非法参数：表示输入参数不合法。

类型不匹配：表示参数的数据类型不一致。

7.2.4.12 张量取倒数操作

7.2.4.12.1 功能

计算输入张量的倒数（逐元素求倒数值）。

7.2.4.12.2 接口参数

张量取倒数操作函数前向接口应符合表66，C语言示例见A.2.4.12。

表 66 张量取倒数操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输出张量	输出	必选	表示计算结果

7.2.4.12.3 接口返回值

没有错误：操作成功。
非法参数：表示输入参数不合法。
类型不匹配：表示参数的数据类型不一致。

7.2.4.13 张量对角线元素求和操作

7.2.4.13.1 功能

计算指定2维平面上的对角线元素之和。

7.2.4.13.2 前向接口参数

张量对角线元素求和操作函数前向接口应符合表67，C语言示例见A.2.4.13。

表 67 张量对角线元素求和操作函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
对角线的偏移位置	输入	可选	二维平面中对角线的位置，即相对主对角线的偏移。0代表主对角线，负数代表主对角线左下的对角线，正数代表主对角线右上的对角线，默认为0
二维平面的第一维	输入	可选	对角线所在二维平面的第一维
二维平面的第二维	输入	可选	对角线所在二维平面的第二维
输出张量	输出	必选	表示计算结果，数据类型与输入张量相同

7.2.4.13.3 前向接口返回值

没有错误：操作成功。
非法参数：输入张量的维数小于2，或设置的轴超出了输入张量的维数。

7.2.4.13.4 后向接口参数

张量对角线元素求和操作函数后向接口应符合表68，C语言示例见A.2.4.13。

表 68　张量对角线元素求和操作函数后向接口参数列表

参数名	类型	可选/必选	描述
输出梯度	输入	必选	表示输出张量的梯度张量
对角线的偏移位置	输入	可选	二维平面中对角线的位置，即相对主对角线的偏移。0代表主对角线，负数代表主对角线左下的对角线，正数代表主对角线右上的对角线
二维平面的第一维	输入	可选	对角线所在二维平面的第一维
二维平面的第二维	输入	可选	对角线所在二维平面的第二维
输入梯度	输出	必选	表示输入张量的梯度张量

7.2.4.13.5　后向接口返回值

没有错误：操作成功。
非法参数：输入张量的维数小于2，或设置的轴超出了输入张量的维数。

7.2.5　比较操作

7.2.5.1　判断张量是否相等

7.2.5.1.1　功能

判断两个张量的对应元素的值是否相等，见式（17）。

$$z_i = (x_i == y_i)\ ?\ true : false \quad \text{（17）}$$

式中：
x——第一个输入张量；
y——第二个输入张量；
z——输出张量。

7.2.5.1.2　接口参数

判断张量是否相等函数前向接口应符合表69，C语言示例见A.2.5.1。

表 69　判断张量是否相等函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔等
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.1.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.5.2　判断张量是否不等

7.2.5.2.1　功能

判断两个张量的对应元素的值是否不相等，见式（18）。

$$z_i = x_i\,!=y_i\,?\,true:false \quad \text{（18）}$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.5.2.2 接口参数

断张量是否不等函数前向接口应符合表70，C语言示例见A.2.5.2。

表 70 判断张量是否不等函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔等
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.2.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.5.3 判断张量是否大于

7.2.5.3.1 功能

判断第一个张量的值是否大于第二个张量的对应元素，见式（19）。

$$z_i = x_i > y_i\,?\,true:false \quad \text{（19）}$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.5.3.2 接口参数

判断张量是否大于函数前向接口应符合表71，C语言示例见A.2.5.3。

表 71 判断张量是否大于函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.3.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.5.4 判断张量是否大于等于

7.2.5.4.1 功能

判断第一个张量元素值是否大于等于第二个张量的对应元素，见式（20）。

$$z_i = x_i \geq y_i \ ? \ true : false \quad \text{（20）}$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.5.4.2 接口参数

判断张量是否大于等于函数前向接口应符合表72，C语言示例见A.2.5.4。

表 72 判断张量是否大于等于函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.4.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.5.5 判断张量是否小于

7.2.5.5.1 功能

判断第一个张量的元素值是否小于第二个张量的对应元素值，见式（21）。

$$z_i = x_i < y_i \ ? \ true : false \quad \text{（21）}$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.5.5.2 接口参数

判断张量是否小于函数前向接口应符合表73，C语言示例见A.2.5.5。

表 73 判断张量是否小于函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.5.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.5.6 判断张量是否小于等于

7.2.5.6.1 功能

判断第一个张量的元素值是否小于等于第二个张量的对应元素值，见式（22）。

$$z_i = x_i \le y_i \,?\, true : false \quad (22)$$

式中：

x——第一个输入张量；

y——第二个输入张量；

z——输出张量。

7.2.5.6.2 接口参数

判断张量是否小于等于函数前向接口应符合表74，C语言示例见A.2.5.6。

表 74 判断张量是否小于等于函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，元素类型为布尔

7.2.5.6.3 接口返回值

没有错误：操作成功。

类型不匹配：表示参数的数据类型不一致。

7.2.5.7 判断张量是否值相近

7.2.5.7.1 功能

逐个检查输入张量与对比张量的所有元素，当两个待比较张量的所有元素均在一定容忍误差范围内，则认为这两个张量是相等的。比较公式见式（23）。

$$|input - other| \le atol + rtol \times |other| \quad (23)$$

式中：

input——输入张量；

$other$——对比张量；

atol——绝对误差；

rtol——相对误差；

$|*|$——表示取绝对值。

7.2.5.7.2 接口参数

判断张量是否值相近函数前向接口应符合表75，C语言示例见A.2.5.7。

表 75　判断张量是否值相近函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
对比张量	输入	必选	与第一个输入张量在计算上兼容
相对容忍误差	输入	可选	元素类型可以为整数、浮点实数，默认为1e-5
绝对容忍误差	输入	可选	元素类型可以为整数、浮点实数，默认为1e-8
NaN判断标识	输入	可选	与第一个输入张量在计算上兼容,为bool类型，默认为false
输出张量	输出	必选	表示计算结果，为布尔类型

7.2.5.7.3　接口返回值

没有错误：操作成功。
类型不匹配：张量的数据类型不一致。
非法参数：张量形状不匹配。

7.2.6　逻辑操作

7.2.6.1　张量的“逻辑与”操作

7.2.6.1.1　功能

计算两个输入张量的逐元素与操作。

7.2.6.1.2　接口参数

张量的“逻辑与”操作函数前向接口应符合表76，C语言示例见A.2.6.1。

表 76　张量的“逻辑与”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔类型，零值将被当作false， 非零值被当作true
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，为布尔类型

7.2.6.1.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.6.2　张量的“逻辑或”操作

7.2.6.2.1　功能

计算两个输入张量的逐元素或操作。

7.2.6.2.2　接口参数

张量的“逻辑或”操作函数前向接口应符合表77，C语言示例见A.2.6.2。

表 77　张量的“逻辑或”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔类型，零值将被当作false，非零值被当作true
输出张量	输出	必选	表示计算结果，为布尔类型

7.2.6.2.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.6.3　张量的“逻辑非”操作

7.2.6.3.1　功能

逐元素地对一个输入张量进行逻辑非运算。

7.2.6.3.2　接口参数

张量的“逻辑非”操作函数前向接口应符合表78，C语言示例见A.2.6.3。

表 78　张量的“逻辑非”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔类型，零值将被当作false，非零值被当作true
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，为布尔类型

7.2.6.3.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.6.4　张量的“逻辑异或”操作

7.2.6.4.1　功能

计算两个输入张量的逐元素异或操作。

7.2.6.4.2　接口参数

张量的“逻辑异或”操作函数前向接口应符合表79，C语言示例见A.2.6.4。

表 79　张量的“逻辑异或”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数、布尔类型，零值将被当作false，非零值被当作true
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	表示计算结果，为布尔类型

7.2.6.4.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7 位操作

7.2.7.1 逐位“与”操作

7.2.7.1.1 功能

输出张量的每一个元素为输入张量对应元素的逐位与的值。

7.2.7.1.2 接口参数

逐位“与”操作函数前向接口应符合表80，C语言示例见A.2.7.1。

表 80 逐位“与”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	输出张量

7.2.7.1.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7.2 逐位“或”操作

7.2.7.2.1 功能

输出张量的每一个元素为输入张量对应元素的逐位或的值。

7.2.7.2.2 接口参数

逐位“或”操作函数前向接口应符合表81，C语言示例见A.2.7.2。

表 81 逐位“或”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	输出张量

7.2.7.2.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7.3 逐位“异或”操作

7.2.7.3.1 功能

输出张量的每一个元素为输入张量对应元素的逐位异或的值。

7.2.7.3.2 接口参数

逐位“异或”操作函数前向接口应符合表82，C语言示例见A.2.7.3。

表 82 逐位“异或”操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输入张量	输入	必选	与第一个输入张量在计算上兼容
输出张量	输出	必选	输出张量

7.2.7.3.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7.4 逐位反转操作

7.2.7.4.1 功能

逐元素地对输入张量进行按位取反的运算。

7.2.7.4.2 接口参数

逐位反转操作函数前向接口应符合表83，C语言示例见A.2.7.4。

表 83 逐位反转操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输出张量	输出	必选	输出张量

7.2.7.4.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7.5 逐位左移操作

7.2.7.5.1 功能

逐元素地对输入张量进行按位左移运算。

7.2.7.5.2 接口参数

逐位左移操作函数前向接口应符合表84，C语言示例见A.2.7.5。

表 84　逐位左移操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输出张量	输出	必选	输出张量

7.2.7.5.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.7.6　逐位右移操作

7.2.7.6.1　功能

逐元素地对输入张量进行按位右移运算。

7.2.7.6.2　接口参数

逐位右移操作函数前向接口应符合表85，C语言示例见A.2.7.6。

表 85　逐位右移操作函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、布尔类型
输出张量	输出	必选	输出张量

7.2.7.6.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.8　幂操作

7.2.8.1　幂

7.2.8.1.1　功能

计算每个元素的指数值，见式（24）。

$$y_i = x_i^{e_i} \quad \text{（24）}$$

式中：
x——输入张量；
e——输入幂张量；
y——输出张量。
注意：x和e的形状必须相等。

7.2.8.1.2　接口参数

幂函数前向接口应符合表86，C语言示例见A.2.8.1。

表 86　幂函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
幂张量	输入	必选	表示输入张量对应元素的幂
输出张量	输出	必选	表示计算结果

7.2.8.1.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.8.2　平方根

7.2.8.2.1　功能

计算输入张量每个元素的平方根，见式（25）。

$$y_i = \sqrt{x_i} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (25)$$

式中：
x——输入张量；
y——输出张量；
√——表示取平方根。

7.2.8.2.2　接口参数

平方根函数前向接口应符合表87，C语言示例见A.2.8.2。

表 87　平方根函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.8.2.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数出错。

7.2.8.3　平方根倒数

7.2.8.3.1　功能

计算输入张量每个元素平方根的倒数，见式（26）。

$$y_i = {1}\Big/{\sqrt{x_i}} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (26)$$

式中：
x——输入张量；
y——输出张量；
√——表示取平方根。

7.2.8.3.2 接口参数

平方根倒数函数前向接口应符合表88，C语言示例见A.2.8.3。

表 88 平方根倒数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.8.3.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数出错。

7.2.8.4 平方数

7.2.8.4.1 功能

计算输入张量每个元素的平方数，见式（27）。

$$y_i = x_i^2 \quad \cdots\cdots (27)$$

式中：
x——输入张量；
y——输出张量。

7.2.8.4.2 接口参数

平方数函数前向接口应符合表89，C语言示例见A.2.8.4。

表 89 平方数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.8.4.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.9 舍入操作

7.2.9.1 向下取整

7.2.9.1.1 功能

输出张量的每一个元素是输入张量的每一个元素的下取整的值，见式（28）。

$$y_i = \lfloor x_i \rfloor \quad \cdots\cdots (28)$$

式中：
x——输入张量；

y——输出张量。

7.2.9.1.2 接口参数

向下取整函数前向接口应符合表90，C语言示例见A.2.9.1。

表 90 向下取整函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示取整的结果

7.2.9.1.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.9.2 向上取整

7.2.9.2.1 功能

输出张量的每一个元素是输入张量的每一个元素的上取整的值，见式（29）。

$$y_i = \lceil x_i \rceil \quad \text{（29）}$$

式中：
x——输入张量；
y——输出张量。

7.2.9.2.2 接口参数

向上取整函数前向接口应符合表91，C语言示例见A.2.9.2。

表 91 向上取整函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示取整的结果

7.2.9.2.3 接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.9.3 截断取整

7.2.9.3.1 功能

输出张量的每一个元素是输入张量的每一个元素的值截尾取整，即采用round to zero模式。

7.2.9.3.2 接口参数

截断取整函数前向接口应符合表92，C语言示例见A.2.9.3。

表 92　截断取整函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示取整的结果

7.2.9.3.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.9.4　就近取整

7.2.9.4.1　功能

输出张量的每一个元素四舍五入后的值，采用round to nearest模式。

7.2.9.4.2　接口参数

就近取整函数前向接口应符合表93，C语言示例见A.2.9.4。

表 93　就近取整函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示取整的结果

7.2.9.4.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.9.5　通用舍入取整

7.2.9.5.1　功能

输出张量的每一个元素是输入张量的每一个元素的取整的值，并通过取整模式参数选择舍入模式。

7.2.9.5.2　接口参数

通用舍入取整函数前向接口应符合表94，C语言示例见A.2.9.5。

表 94　通用舍入取整函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
取整模式	输入	必选	表示取整方式
输出张量	输出	必选	元素类型可以为浮点实数

7.2.9.5.3　接口返回值

没有错误：操作成功。
类型不匹配：表示参数的数据类型不一致。

7.2.10 三角函数

7.2.10.1 正弦函数

7.2.10.1.1 功能描述：

计算输入张量每个元素的sin值，见式（30）。

$$y_i = \sin(x_i) \quad (30)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.1.2 接口参数

正弦函数前向接口应符合表95，C语言示例见A.2.10.1。

表95 正弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.1.3 接口返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。

类型不匹配：表示参数的数据类型不一致。

7.2.10.2 余弦函数

7.2.10.2.1 功能

计算输入张量每个元素的cos值，见式（31）。

$$y_i = \cos(x_i) \quad (31)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.2.2 接口参数

余弦函数前向接口应符合表96，C语言示例见A.2.10.2。

表96 余弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.2.3 接口返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。

类型不匹配：表示参数的数据类型不一致。

7.2.10.3 正切函数

7.2.10.3.1 功能

计算输入张量每个元素的tan值，见式（32）。

$$y_i = \tan(x_i) \quad (32)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.3.2 接口参数

正切函数前向接口应符合表97，C语言示例见A.2.10.3。

表97 正切函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.3.3 接口返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。

类型不匹配：表示参数的数据类型不一致。

7.2.10.4 反正弦函数

7.2.10.4.1 功能

计算输入张量每个元素的asin值，见式（33）。

$$y_i = \operatorname{asin}(x_i) \quad (33)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.4.2 接口参数

反正弦函数前向接口应符合表98，C语言示例见A.2.10.4。

表98 反正弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.4.3 函数返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。

类型不匹配：表示参数的数据类型不一致。

7.2.10.5 反余弦函数

7.2.10.5.1 功能

计算输入张量每个元素的acos值，见式（34）。

$$y_i = \mathrm{acos}(x_i) \quad \cdots\cdots (34)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.5.2 接口参数

反余弦函数前向接口应符合表99，C语言示例见A.2.10.5。

表 99 反余弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.5.3 接口返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。

类型不匹配：表示参数的数据类型不一致。

7.2.10.6 反正切函数

7.2.10.6.1 功能

计算输入张量每个元素的atan值，见式（35）。

$$y_i = \mathrm{atan}(x_i) \quad \cdots\cdots (35)$$

式中：

x——输入张量；

y——输出张量；

7.2.10.6.2 接口参数

反正切函数前向接口应符合表100，C语言示例见A.2.10.6。

表 100 反正切函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.10.6.3 接口返回值

没有错误：操作成功。

内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数不合法。

7.2.11 双曲函数

7.2.11.1 双曲正弦函数

7.2.11.1.1 功能

计算输入张量每个元素的sinh值，见式（36）。

$$y_i = \sinh(x_i) \quad (36)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.1.2 接口参数

双曲正弦函数前向接口应符合表101，C语言示例见A.2.11.1。

表 101 双曲正弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.1.3 接口返回值

没有错误：操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.11.2 双曲余弦函数

7.2.11.2.1 功能

计算输入张量每个元素的cosh值，见式（37）。

$$y_i = \cosh(x_i) \quad (37)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.2.2 接口参数

双曲余弦函数前向接口应符合表102，C语言示例见A.2.11.2。

表 102 双曲余弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.2.3 接口返回值

没有错误：操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.11.3 双曲正切函数

7.2.11.3.1 功能

计算输入张量每个元素的tanh值，见式（38）。

$$y_i = \tanh(x_i) \quad (38)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.3.2 接口参数

双曲正切函数前向接口应符合表103，C语言示例见A.2.11.3。

表 103 双曲正切函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.3.3 接口返回值

没有错误：操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.11.4 反双曲正弦函数

7.2.11.4.1 功能

计算输入张量每个元素的asinh值，见式（39）。

$$y_i = \operatorname{asinh}(x_i) \quad (39)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.4.2 接口参数

反双曲正弦函数前向接口应符合表104，C语言示例见A.2.11.4。

表 104 反双曲正弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.4.3 接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.11.5 反双曲余弦函数

7.2.11.5.1 功能

计算输入张量每个元素的acosh值，见式（40）。

$$y_i = \operatorname{acosh}(x_i) \quad (40)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.5.2 接口参数

反双曲余弦函数前向接口应符合表105，C语言示例见A.2.11.5。

表 105 反双曲余弦函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.5.3 接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数不合法。

7.2.11.6 反双曲正切函数

7.2.11.6.1 功能描述

计算输入张量每个元素的atanh值，见式（41）。

$$y_i = \operatorname{atanh}(x_i) \quad (41)$$

式中：
x——输入张量；
y——输出张量。

7.2.11.6.2 接口参数

反双曲正切函数前向接口应符合表106，C语言示例见A.2.11.6。

表 106 反双曲正切函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.11.6.3 接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数不合法。

7.2.12 指对函数

7.2.12.1 指数函数

7.2.12.1.1 功能

计算输入张量每个元素的指数值，见式（42）。

$$y_i = e^{x_i} \quad (42)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.1.2 接口参数

指数函数前向接口应符合表107，C语言示例见A.2.12.1。

表 107 指数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.1.3 接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.12.2 指数函数扩展

7.2.12.2.1 功能

主元素按式（43）计算。

$$y_i = e^{x_i} - 1 \quad (43)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.2.2 接口参数

指数函数扩展前向接口应符合表108，C语言示例见A.2.12.2。

表 108　指数函数扩展函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.2.3　函数返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。

7.2.12.3　以 e 为底的对数函数

7.2.12.3.1　功能

计算入张量每个元素的指数值，见式（44）。

$$y_i = \log_e(x_i) \quad (44)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.3.2　接口参数

以e为底的对数函数前向接口应符合表109，C语言示例见A.2.12.3。

表 109　以 e 为底的对数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.3.3　接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致
非法参数：表示参数不合法。

7.2.12.4　以 e 为底的对数函数扩展

7.2.12.4.1　功能

计算输入张量每个元素的指数值，见式（45）。

$$y_i = \log_e(1 + x_i) \quad (45)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.4.2　接口参数

以e为底的对数函数前向接口应符合表110，C语言示例见A.2.12.4。

表 110　以 e 为底的对数函数扩展函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.4.3　接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致
非法参数：表示参数不合法。

7.2.12.5　以 10 为底的对数函数

7.2.12.5.1　功能

计算输入张量每个元素的指数值，见式（46）。

$$y_i = \log_{10}(x_i) \quad \cdots\cdots (46)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.5.2　接口参数

以10为底的对数函数前向接口应符合表111，C语言示例见A.2.12.5。

表 111　以 10 为底的对数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.5.3　接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致
非法参数：表示参数不合法。

7.2.12.6　以 2 为底的对数函数

7.2.12.6.1　功能

计算输入张量每个元素的指数值，见式（47）。

$$y_i = \log_2(x_i) \quad \cdots\cdots (47)$$

式中：
x——输入张量；
y——输出张量。

7.2.12.6.2 接口参数

以2为底的对数函数前向接口应符合表112，C语言示例见A.2.12.6。

表 112 以 2 为底的对数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为浮点实数
输出张量	输出	必选	表示计算结果

7.2.12.6.3 接口返回值

没有错误：表示操作成功。
内存不足：表示输出向量分配空间不足。
类型不匹配：表示参数的数据类型不一致。
非法参数：表示参数不合法。

7.2.13 规约类操作

7.2.13.1 规约

7.2.13.1.1 功能

计算一个张量的各个维度上元素的规约操作。

7.2.13.1.2 前向接口参数

规约函数前向接口应符合表113，C语言示例见A.2.13.1。

表 113 规约函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等
规约类型	输入	可选	表示规约的类型，包括求和、乘积、最大、最小、均值、逻辑与、逻辑或、逻辑异或、按位与、按位或、按位异或等
规约维度数组	输入	可选	axis数组的首地址，axis数组表示需要规约的维度，其元素类型为int，各元素取值范围是[0，rank(输入张量)]。当其值为NULL时，则规约所有维度；当axis数组值为{k}，表示规约第k维度；当axis数组值为{P，...，Q}表示同时规约P、...、Q维度。默认值：NULL。
规约维度数组长度	输入	可选	归约维度数组元素个数。默认值：0
维度保留	输入	可选	对指定的维度进行规约后，是否保留相应的维度，若保留则规约维度大小为1。默认值：false。
输出张量	输出	必选	表示规约操作的计算结果。

7.2.13.1.3 前向接口返回值

没有错误：表示操作成功。
对象未初始化：表示张量对象未初始化。
类型不匹配：表示输入张量对象的类型和输出规约张量的类型不匹配。

内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.13.1.4 后向接口参数

规约函数后向接口应符合表114，C语言示例见A.2.13.1。

表 114 规约函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是归约操作的计算结果
输入张量	输入	可选	元素类型可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等。即前向接口中输入张量
规约类型	输入	可选	表示规约的类型，包括求和、乘积、最大、最小、均值、逻辑与、逻辑或、逻辑异或、按位与、按位或、按位异或等。即前向接口中归约类型。默认值：求和
规约维度数组首地址	输入	可选	axis数组的首地址，axis数组表示需要规约的维度，其元素类型为int，各元素取值范围是[0，rank(输入张量)]。当其值为NULL时，则规约所有维度；当axis数组值为{k}，表示规约第k维度；当axis数组值为{P，...，Q}表示同时规约P、...、Q维度。即前向接口中归约维度数组首地址。默认值：NULL
规约维度数组长度	输入	可选	归约维度数组元素个数。前向接口中规约维度数组长度。默认值：0
维度保留	输入	可选	对指定的维度进行规约后，是否保留相应的维度，若保留则规约维度大小为1。即前向接口中维度保留值。默认值：false
输入张量梯度	输出	必选	表示前向接口中输入张量的梯度。输入张量元素类型可以为有符号整数、无符号整数、浮点实数、浮点复数、布尔等

7.2.13.1.5 后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示张量对象未初始化。
类型不匹配：表示输入张量对象的类型和输出规约张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.13.2 前缀和

7.2.13.2.1 功能

沿给定的轴计算输入张量的累加和。

7.2.13.2.2 前向接口参数

前缀和函数前向接口应符合表115，C语言示例见A.2.13.2。

表 115　前缀和函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为整数、浮点数
累加的维度	输入	可选	输入张量累加的维度，-1代表按照1维张量求全局的前缀和。默认值：-1
输出张量	输出	必选	元素数据类型由数据类型参数指定，或与输入张量相同

7.2.13.2.3　前向接口返回值

没有错误：表示操作成功。
类型不匹配：输入张量对象的类型和dtype计算不兼容。
超出范围：累加的维度超出输入张量维度。
非法参数：其它参数不合法。

7.2.13.2.4　后向接口参数

前缀和函数后向接口应符合表116，C语言示例见A.2.13.2。

表 116　前缀和函数后向接口函数参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量元素数据类型由数据类型参数指定，或与输入张量相同
累加的维度	输入	可选	表示输入张量累加的维度，-1代表按照1维张量求全局的前缀和。即前向接口中的累加维度。默认值：-1
输入张量梯度	输出	必选	表示前向接口中输入张量的梯度。输入张量元素类型可以为整数、浮点数

7.2.13.2.5　后向接口返回值

没有错误：操作成功。
类型不匹配：张量的数据类型不一致。
非法参数：累加的轴超出了输入张量的维数。

7.2.14　索引操作

7.2.14.1　最大索引

7.2.14.1.1　功能

计算张量在指定维度上最大元素的索引。

7.2.14.1.2　接口参数

最大索引函数前向接口应符合表117，C语言示例见A.2.14.1。

表 117　最大索引函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	待求解最大索引的张量，元素类型可以为有整数、浮点数
索引维度	输入	可选	表示求最大元素索引的维度。k>=0表示第k+1维度；k<0表示倒数第k维度。默认值：-1
保持维度	输入	可选	表示是否在输出张量中保留减小的维度。如果 keepdim 为true，则输出张量和输入张量具有相同的维度(减少的维度除外，减少的维度的大小为1)。默认值：false。
输出数据类型	输入	可选	表示输出Tensor的数据类型。默认值：int64
输出张量	输出	必选	表示索引结果的张量

7.2.14.1.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
超出范围：表示索引维度超出张量维度。

7.2.14.2　最小索引

7.2.14.2.1　功能

计算张量在指定维度上最小元素的索引。

7.2.14.2.2　接口参数

最小索引函数前向接口应符合表118，C语言示例见A.2.14.2。

表 118　最小索引函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	待求解最小索引的张量，元素类型可以为有整数、浮点数
索引维度	输入	可选	表示求最小元素索引的维度。k>=0表示第k+1维度；k<0表示倒数第k维度。默认值：-1
保持维度	输入	可选	表示是否在输出张量中保留减小的维度。如果 keepdim 为true，则输出张量和输入张量具有相同的维度(减少的维度除外，减少的维度的大小为1)。默认值：false。
输出数据类型	输入	可选	表示输出Tensor的数据类型。默认值：int64
输出张量	输出	必选	表示索引结果的张量

7.2.14.2.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
超出范围：表示索引维度超出张量维度。

7.2.14.3　排序索引

7.2.14.3.1　功能

对输入张量沿索引轴进行排序，输出排序后的数据和相应的位置索引。

7.2.14.3.2 前向接口参数

排序索引函数前向接口应符合表119，C语言示例见A.2.14.3。

表 119 排序索引函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为有整数、浮点数
排序维度	输入	可选	表示待排序的维度，k>=0表示第k+1维度；k<0表示倒数第k维度。默认值：-1
排序方式	输入	可选	布尔类型，true表示算法按照降序排序，否则按照升序排序。默认值为true
输出张量	输出	必选	表示排序后的张量，形状、数据类型与输入张量相同
索引张量	输出	必选	表示索引结果的张量

7.2.14.3.3 前向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
超出范围：表示排序维度超出张量维度。

7.2.14.3.4 后向接口参数

排序索引函数后向接口应符合表120，C语言示例见A.2.14.3。

表 120 排序索引函数后向接口函数参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是排序后的张量，形状、数据类型与输入张量相同
索引张量	输入	必选	表示索引结果的张量。即前向接口中的索引张量
输入张量梯度	输出	必选	表示前向接口中的输入张量的梯度。输入张量元素类型可以为有整数、浮点数

7.2.14.3.5 后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
超出范围：表示排序维度超出张量维度。

7.2.14.4 Top K索引

7.2.14.4.1 功能

沿最后一维查找输入张量的前 K 个最大项，返回最大项的值和索引。若输入张量的形状为$[N0,\dots,Nd-2,Nd-1]$，那么输出张量和索引张量的形状为$[N0,\dots,Nd-2,K]$。

7.2.14.4.2 接口参数

Top K索引函数前向接口应符合表121，C语言示例见A.2.14.4。

表 121 Top K 索引函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	待索引张量，元素类型可以为有整数、浮点数
K值	输入	必选	表示寻找的最大前K项，值须大于0且小于输入张量最后一维的大小
索引维度	输入	可选	表示TopK取数据的维度。假如P>=0，Q>0,那么-Q表示倒数第Q维度，P表示第P+1维度。默认值：-1
排序方式	输入	可选	布尔类型，true表示算法按照降序的算法排序，否则按照升序排序。默认值为true
有序返回	输入	可选	布尔类型，控制返回的k个结果是否严格按照有序返回。默认值：true
输出张量	输出	必选	表示排序后的张量，数据类型与输入张量相同
索引张量	输出	必选	表示索引结果的张量

7.2.14.4.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
非法参数：表示其他参数不合法。

7.2.14.5 非零索引

7.2.14.5.1 功能描述

返回输入张量中非零元素的坐标。

7.2.14.5.2 接口参数

非零索引函数前向接口应符合表122，C语言示例见A.2.14.5。

表 122 非零索引函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	元素类型可以为有整数、浮点数
输出格式	输入	可选	表示输出格式是否是一维张量构成的元组格式，类型为布尔。默认值：true
输出张量	输出	必选	表示索引结果的张量

7.2.14.5.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象没有初始化。
超出范围：表示不存在非零元素。
内存不足：表示输出向量分配空间不足
非法参数：表示其他参数不合法。
其它内部错误：内部调用操作出错。

7.2.15 复数操作

7.2.15.1 复数构建

7.2.15.1.1 功能

将两个元素类型为浮点实数的张量转换为元素类型为浮点复数的张量。

7.2.15.1.2 接口参数

复数构建函数前向接口应符合表123，C语言示例见A.2.15.1。

表 123 复数构建函数参数列表

参数名	类型	可选/必选	描述
实部	输入	必选	表示需要转换成复数张量的实部，元素类型为浮点实数。
虚部	输入	必选	表示需要转换成复数张量的虚部，元素类型为浮点实数。
复数张量	输出	必选	表示复数张量

7.2.15.1.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
类型不匹配：表示输入张量对象的类型和要转换的类型不兼容、两个输入张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
其它内部错误：表示内部调用出错。

7.2.15.2 复数共轭

7.2.15.2.1 功能

求取复数共轭。

7.2.15.2.2 接口参数

复数共轭函数前向接口应符合表124，C语言示例见A.2.15.2。

表 124 复数共轭函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示要求取复数共轭的输入张量
输出张量	输出	必选	表示共轭结果，若输入张量元素是实数，则输出张量等于输入张量

7.2.15.2.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
类型不匹配：表示输入张量对象的类型和要转换的类型不兼容。
内存不足：表示输出张量分配空间不足。
其它内部错误：表示内部调用出错。

7.2.15.3 获取虚部

7.2.15.3.1 功能

获取输入张量的虚部。

7.2.15.3.2 接口参数

获取虚部函数前向接口应符合表125，C语言示例见A.2.15.3。

表 125 获取虚部函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示要求取虚部的输入张量
输出张量	输出	必选	表示获取输入张量的虚部，如果输入张量的元素是实数，则输出张量为0

7.2.15.3.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
类型不匹配：表示输入张量对象的类型和要转换的类型不兼容。
内存不足：表示输出张量分配空间不足。
其它内部错误：表示内部调用出错。

7.2.15.4 获取实部

7.2.15.4.1 功能

获取输入张量的实部。

7.2.15.4.2 接口参数

获取实部函数前向接口应符合表126，C语言示例见A.2.15.4。

表 126 获取实部函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示要求取实部的输入张量
输出张量	输出	必选	表示获取输入张量的实部，如果输入张量的元素是实数，则输出张量等于输入张量

7.2.15.4.3 接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
类型不匹配：表示输入张量对象的类型和要转换的类型不兼容。
内存不足：表示输出张量分配空间不足。
其它内部错误：表示内部调用出错。

7.2.16 信号处理类

7.2.16.1 复数到复数的快速傅里叶变换

7.2.16.1.1 功能

沿着指定维度对输入张量进行复数到复数的快速离散傅里叶变换。

7.2.16.1.2 前向接口参数

快速傅里叶变换函数前向接口应符合表127，C语言示例见A.2.16.1。

表 127　前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	需要进行傅里叶变换的输入张量，数据类型为浮点复数
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。默认值：0
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放、除以N的平方根、除以N。默认值：不缩放
变换方式	输入	可选	为true表示正变换，为false表示逆变换。默认值：true
输出张量	输入 输出	必选	表示快速逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，数据类型为浮点复数

7.2.16.1.3　前向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.16.1.4　后向接口参数

快速傅里叶变换函数前向接口应符合表128，C语言示例见A.2.16.1。

表 128　后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口输出张量的梯度。数据类型为浮点复数。输入张量是进行傅里叶变换的输入张量，数据类型为浮点复数
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。即前向接口中的维度数组首地址。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。即前向接口中的维度数组长度。默认值：0
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放、除以N的平方根、除以N。即前向接口中的缩放模式。默认值：不缩放
变换方式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放、除以N的平方根、除以N。即前向接口中的变换方式。默认值：true
输入张量梯度	输出	必选	输入张量的梯度。数据类型为浮点复数。输入张量是进行傅里叶变换的输入张量，数据类型为浮点复数

7.2.16.1.5　后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.16.2 实数到复数的快速逆傅里叶变换

7.2.16.2.1 功能

沿着指定维度对输入张量进行实数到复数的快速离散傅里叶变换。

7.2.16.2.2 前向接口参数

快速逆傅里叶变换函数前向接口应符合表129，C语言示例见A.2.16.2。

表 129 前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	需要进行傅里叶变换的输入张量，数据类型为浮点实数
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。默认值：0
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放、除以N的平方根、除以N。默认值：不缩放
变换方式	输入	可选	为true表示正变换，为false表示逆变换。默认值：true
复数结果的保留方式	输入	可选	为true表示不保留共轭结果，正向和后向都可以节省1半的存储空间。为false表示保留全部结果。默认值：true
输出张量	输入 输出	必选	表示快速逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，元素类型为浮点复数

7.2.16.2.3 前向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.16.2.4 后向接口参数

快速逆傅里叶变换函数前向接口应符合表130，C语言示例见A.2.16.2。

表 130 后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度，数据类型为浮点复数。输出张量是快速逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，元素类型为浮点复数
输入张量	输入	必选	需要进行傅里叶变换的输入张量，数据类型为浮点实数。即前向接口中的输入张量，因为如果正向计算的复数结果不保留共轭，则反向计算逻辑中需要知道输入张量
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。即前向接口中的维度数组首地址。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。即前向接口中的维度数组长度。默认值：0
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放、除以N的平方根、除以N。即前向接口中的缩放模式。默认值：不缩放
变换方式	输入	可选	为true表示正变换，为false表示逆变换。前向接口中的变换方式。默认值：true
输入张量梯度	输出	必选	输入张量的梯度，数据类型为浮点实数。输入张量是需要进行傅里叶变换的输入张量，数据类型为浮点实数

7.2.16.2.5 后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用出错。

7.2.16.3 复数到实数的快速傅里叶变换

7.2.16.3.1 功能

沿着指定维度对输入张量进行复数到实数的快速离散傅里叶变换。

7.2.16.3.2 前向接口参数

快速逆傅里叶变换函数前向接口应符合表131，C语言示例见A.2.16.2。

表 131 前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	需要进行傅里叶变换的输入张量，数据类型为浮点复数
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。默认值：0

表 131 前向接口参数列表（续）

参数名	类型	可选/必选	描述
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放，除以N的平方根，除以N。默认值：不缩放
变换方式	输入	可选	为true表示正变换，为false表示逆变换。默认值：true
输出实数结果的最后一维大小	输入	必选	因为当复数最后1维为N时，可以变换得到最后1维为(N-1)*2 或 (N-1)*2+1 的实数，所以需要指定是得到哪种维度的实数结果
输出张量	输入 输出	必选	表示快速逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，数据类型为浮点实数

7.2.16.3.3 前向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。
非法参数：表示其他参数不合法。
其它内部错误：表示内部的调用操作出错。

7.2.16.3.4 后向接口参数

快速逆傅里叶变换函数前向接口应符合表132，C语言示例见A.2.16.2。

表 132 后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度，数据类型为浮点实数。输出张量是快速逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，数据类型为浮点实数
维度数组首地址	输入	可选	进行傅里叶变换所沿着的维度数组的首地址。维度数组表示要在哪些维度进行多维傅里叶变换；首地址为NULL表示在最后一个维度进行傅里叶变换。即前向接口中的维度数组首地址。默认值：NULL
维度数组长度	输入	可选	维度数组的元素个数。即前向接口中的维度数组长度。默认值：0
缩放模式	输入	可选	设输入张量中总的元素有N个，缩放模式有3种：不缩放，除以N的平方根，除以N。即前向接口中的缩放模式。默认值：不缩放
变换方式	输入	可选	为true表示正变换，为false表示逆变换。即前向接口中的变换方式。默认值：true
输入张量梯度	输出	必选	表示前向接口中输入张量的梯度，数据类型为浮点复数。输入张量是需要进行傅里叶变换的输入张量，数据类型为浮点复数。

7.2.16.3.5 后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量没有初始化。
类型不匹配：表示输入张量对象的类型和输出张量的类型不匹配。
内存不足：表示输出张量分配空间不足。

非法参数：表示其他参数不合法。

其它内部错误：表示内部的调用操作出错。

7.2.17 线性代数操作

7.2.17.1 矩阵乘法

7.2.17.1.1 功能

计算两个张量的乘积。具体操作过程依据张量的维度分为如下若干情况：

——如果两个输入张量都是一维张量，则执行向量点积操作；

——如果两个输入张量都是二维张量，则执行矩阵乘法操作；

——如果第一个输入张量是一维张量，第二个输入张量是二维张量，则将第一个输入张量扩展成二维张量（在形状数组的起始位置插入元素“1”），然后执行矩阵乘法操作，并在最终的输出结果中去除添加的维度；

——如果第一个输入参数是二维张量，第二个输入参数是一维张量，则执行矩阵向量乘操作；

——如果其中一个输入张量维度大于2,则执行批量矩阵乘法操作。a)当第一个输入张量维度是1时，则在该张量的维度之前再添加一个为1的维度（形状数组的起始位置插入元素“1”），将第二个张量的高维看作batch数量，以计算批量矩阵运算，计算结束后，将添加的维度去除；b)当第二个输入张量维度是1时，则在该张量的维度之后添加一个为1的维度（形状数组的末尾插入元素“1”），将第一个张量的高维看作batch数量，以进行批量矩阵运算，运算结束后，将添加的维度去除；c)当两个输入张量的维度高于2时，inner-most的两维用于矩阵运算；其他维度表示batch数量，是可广播的(broadcastable)。

——如果某个输入张量维度低于1，则返回STATUS_DIMENSIONS_MISMATCH错误码。

7.2.17.1.2 接口参数

矩阵乘法函数前向接口应符合表133，C语言示例见A.2.17.1。

表133 矩阵乘法函数参数列表

参数名	类型	可选/必选	描述
输入张量1	输入	必选	描述用于矩阵乘法的第一个输入张量mat1
输入张量2	输入	必选	描述用于矩阵乘法的第二个输入张量mat2
输出张量	输出	必选	输出张量mat1和张量mat2的乘积

7.2.17.1.3 接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

维度不匹配：输入张量维度不匹配等。

7.2.17.2 向量内积

7.2.17.2.1 功能

计算两个张量的内积，具体计算形式按照输入参数有如下描述：

——如果两个输入张量都是一维张量，则计算向量内积；

——如果两个输入张量都是二维张量，则计算矩阵乘法；

——如果某个输入张量是常量，则计算常量与张量乘法；

——如果第二个输入张量是一维张量，则计算第一个输入张量的最后一维与第二个输入张量的内积；

——如果两个输入张量的维度都大于2，则计算第一个输入张量的倒数第一个维度与第二个输入张量的倒数第二个维度的内积。

7.2.17.2.2 前向接口参数

向量内积函数前向接口应符合表134，C语言示例见A.2.17.2。

表 134 向量内积函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量1	输入	必选	描述用于内积的第一个输入张量vec1
输入张量2	输入	必选	描述用于内积的第二个输入张量vec2
输出张量	输出	必选	输出张量vec1和张量vec2的内积

7.2.17.2.3 前向接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.2.4 后向接口参数

向量内积函数后向接口应符合表135，C语言示例见A.2.17.2。

表 135 向量内积函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量梯度是输出张量vec1和张量vec2的内积
输入张量1	输入	必选	用于内积的第一个输入张量vec1，即前向接口中输入张量1
输入张量2	输入	必选	用于内积的第一个输入张量vec2，即前向接口中输入张量2
输入张量1梯度	输出	必选	表示前向接口中输入张量1的梯度。输入张量1是描述用于内积的第一个输入张量vec1

7.2.17.2.5 后向接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.2.6 后向接口参数

向量内积函数后向接口应符合表136，C语言示例见A.2.17.2。

表 136　向量内积函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量梯度是输出张量vec1和张量vec2的内积
输入张量1	输入	必选	用于内积的第一个输入张量vec1，即前向接口中输入张量1
输入张量2	输入	必选	用于内积的第一个输入张量vec2，即前向接口中输入张量2
输入张量2梯度	输出	必选	表示前向接口中输入张量2的梯度。输入张量2是描述用于内积的第一个输入张量vec2

7.2.17.2.7　后向接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
非法参数：表示其他参数出错情况。

7.2.17.3　LU 矩阵分解

7.2.17.3.1　功能

使用部分行主元进行LU分解，见式（48）。

$$mat = P * L * U \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (48)$$

式中：
P——置换矩阵，记录行交换的过程；
L——下三角矩阵，且对角线元素均为1（如果mat的行数大于列数，则为下三角梯形）；
U——上三角矩阵（如果张量mat的列数大于行数，则为上三角梯形）。
mat——（二维）分解矩阵。

7.2.17.3.2　接口参数

LU矩阵分解函数前向接口应符合表137，C语言示例见A.2.17.3。

表 137　LU 矩阵分解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述待分解矩阵，如果维度高于两维，则inner-most的两维用作分解运算，其它高维用于描述批量运算
输出张量	输出	必选	描述分解结果，如果维度高于两维，则inner-most的两维用作描述分解结果，其中下三角为L矩阵（L为对角线为零的下三角矩阵），上三角为U矩阵（U为上三角矩阵），其它高维用于描述批量特征
输出张量	输出	必选	描述主元交换过程的张量

7.2.17.3.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
非法参数：表示其他参数出错情况。

7.2.17.4　Cholesky 矩阵分解

7.2.17.4.1 功能

执行Cholesky分解，见式（49）或式（50）。：

$$mat = U^H * U\ (if\ uplo = 'U') \quad (49)$$

式中：

U——表示上三角矩阵；

mat——对称正定矩阵，复数情况则为Hermitian正定矩阵。

$$mat = L * L^H\ (if\ uplo = 'L') \quad (50)$$

式中：

L——下三角矩阵；

mat——对称正定矩阵，复数情况则为Hermitian正定矩阵。

7.2.17.4.2 接口参数

Cholesky矩阵分解函数前向接口应符合表138，C语言示例见A.2.17.4。

表 138 Cholesky 矩阵分解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示输入张量mat的值是上三角有效还是下三角有效，当为‘U’时，表示输入张量mat存储的上三角的值是有效的，且输出上三角矩阵U，即$mat = U^H * U$；当为‘L’时，表示输入张量mat存储的下三角是有效的，且输出下三角矩阵L，即$mat = L * L^H$
输入张量	输入	必选	描述对称正定矩阵，如果维度高于二维，则inner-most的两维用作分解运算，其它高维用作描述批量计算
输出张量	输出	必选	输出时基于upper的值inner-most的值对应上三角或者下三角矩阵，且其维度与输入矩阵维度保持一致

7.2.17.4.3 接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.5 QR 矩阵分解

7.2.17.5.1 功能

对输入张量进行QR分解，见式（51）。

$$mat = Q * R \quad (51)$$

式中：

Q——正交矩阵；

R——上三角矩阵；

mat——待分解矩阵。

7.2.17.5.2 接口参数

QR矩阵分解函数前向接口应符合表139，C语言示例见A.2.17.5。

表 139　QR 矩阵分解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述待分解矩阵，如果维度高于两维，则inner-most的两维用作矩阵分解运算，其它维度用作描述批量运算
分解模式	输入	可选	表示正交三角分解的行为。假设输出张量计算形状为[*, M, N]、K = min(M, N)：如果mode = "reduced"，则Q形状实际输出为 [*, M, K]、R形状为[*, K, N];如果 mode="complete"，则Q形状实际输出为[*, M, M]、R形状为[*, M, N]; 如果 mode="r"，则不返回Q,只返回 R且形状为[*, K, N]。默认值："reduced"
输出张量	输出	必选	描述输出正交矩阵，如果输入张量维度高于二维，则Q的最内部的两维对应分解结果，其它高维对应输入张量的批量维度
输出张量	输出	可选	描述输出上三角矩阵，其维度特征描述同输出张量Q。分解模式为"r"时无输出

7.2.17.5.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量对象未初始化。
非法参数：表示其他参数出错情况。

7.2.17.6　SVD 奇异值分解

7.2.17.6.1　功能

SVD分解，见式（52）。

$$mat = USV^T \quad (52)$$

式中：
mat——待分解张量；
U——输出张量，大小为m*m；
S——表示分解结果，输出张量大小为n*m；
V——表示分解结果，输出张量大小为n*n；

7.2.17.6.2　接口参数

SVD奇异值分解函数前向接口应符合表140，C语言示例见A.2.17.6。

表 140　SVD 奇异值分解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述待分解矩阵，如果维度高于两维，则inner-most的两维用作矩阵分解运算，其它维度用作描述批量运算
Format选择	输入	可选	布尔型，当format为true时，返回张量U和V仅包含min(n,m)个正交列一维张量，否则返回张量U和V分别包含n和m个正交列一维张量。默认值：true

表 140　SVD 奇异值分解函数参数列表（续）

参数名	类型	可选/必选	描述
输出张量	输出	必选	分解张量U，inner-most的两维描述分解结果，为n × n，当format为true时，包含min(m,n)个正交列一维张量，否则，包含n个正交列一维张量，其它高维对应输入张量的批量特征
输出张量	输出	必选	由奇异值组成的对角张量，其中inner-most的两维描述分解结果，为n × m，其它高维对应输入张量的批量特征
输出张量	输出	必选	分解张量V，其中inner-most的两维描述分解结果，为m × m，当format为true时，包含min(m,n)个正交列一维张量，否则，包含m个正交列一维张量，其它高维对应输入张量的批量特征

7.2.17.6.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量未初始化。
非法参数：表示其他参数出错情况。

7.2.17.7　线性方程组求解

7.2.17.7.1　功能

求解线性方程组，如果adjoint为true，则解X满足方程，见式（53）。

$$\mathrm{mat}\,x = \mathrm{rhs} \quad (53)$$

式中：
x——待估参数向量；
mat——设计向量；
rhs——观测值向量。
如果adjoint为假，则解X满足方程，见式（54）。

$$\mathrm{adjoint}(\mathrm{mat})\,x = \mathrm{rhs} \quad (54)$$

式中：
x——方程组的解向量；
mat——系数向量；
rhs——观测值向量。

7.2.17.7.2　接口参数

线性方程组求解函数前向接口应符合表141，C语言示例见A.2.17.7。

表 141　线性方程组求解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述系数矩阵，如果维度高于两维，则inner-most的两维用作线性方程组求解，其它高维用作描述批量运算，假设用作求解运算的维度为m × m
输入张量	输入	必选	描述线性方程组的右端项，其中inner-most的两维用作描述线性方程组的右端项，假设为m × k，其它高维用作描述批量运算

表 141　线性方程组求解函数参数列表（续）

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述使用系数矩阵mat求解还是使用其伴随矩阵求解(block-wise adjoint matrix)
输出张量	输出	必选	描述输出解张量，其中inner-most的两维用作描述线性方程组的解，为m × k，其它高维用作描述批量运算，且与输入张量描述批量运算的维度相对应

7.2.17.7.3　接口返回值

没有错误：表示操作成功。
对象未初始化：表示输入张量未初始化。
非法参数：其它参数出错情况。

7.2.17.8　最小二乘

7.2.17.8.1　功能

计算方程mat x = rhs的最小二乘问题，即求解解向量x使得Euclidean 2-范数（见式55）最小，线性方程可能是欠定的，唯一确定的或者超定的（即矩阵mat的线性无关的行数小于，等于或者大于线性无关的列的数目）。

$$\|rhs - mat\ x\|^2 \quad (55)$$

式中：
rhs——观测值向量。
mat——系数向量；
x——待估参数向量；

7.2.17.8.2　接口参数

最小二乘求解函数前向接口应符合表142，C语言示例见A.2.17.8。

表 142　最小二乘求解函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述系数矩阵的输入张量，其中inner-most的两维用作求解运算，其它高维用于描述批量运算，假设用于求解运算的维度为m × n
输入张量	输入	必选	系数向量，如果rhs的维度大于等于2，假设其inner-most的维度为m × k，则针对该张量的每一列计算最小二乘运算，其它高维用于描述批量运算
奇异值截止比	输入	可选	描述输入张量mat的奇异值截止比。即如果奇异值比prec与mat的最大奇异值的乘积小，则将该奇异值看作零，默认值：输入张量数据类型对应的机器精度
输出张量	输出	必选	描述输出最小二乘解。如果输入张量rhs为二维的，则解对应输出张量x的k列,另外描述批量计算的维度应与输入张量保持一致。

7.2.17.8.3　接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.9 矩阵求逆

7.2.17.9.1 功能

求解mat张量的逆张量，见式（56）。

$$\mathrm{inv} = mat^{-1} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (56)$$

式中：

mat——输入张量；

inv——输出张量（mat张量的逆）。

7.2.17.9.2 接口参数

矩阵求逆函数前向接口应符合表143，C语言示例见A.2.17.9。

表 143 矩阵求逆函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	描述待求逆的矩阵，如果维度高于两维，则inner-most的两维用作求逆运算，其它高维用作批量运算
输出张量	输出	必选	输入张量的逆张量，如果输入张量mat维度高于两维，则inv的inner-most的两维对应为逆矩阵，其它高维用作描述批量运算

7.2.17.9.3 接口返回值

没有错误：表示操作成功。

类型不匹配：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.10 求特征值及特征向量

7.2.17.10.1 功能

求解mat张量的特征值与特征向量。

7.2.17.10.2 接口参数

求特征值及特征向量函数前向接口应符合表144，C语言示例见A.2.17.10。

表 144 求特征值及特征向量函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	待计算特征值和右端特征向量的方阵，若输入张量mat的维度高于两维，则inner-most的两维用作描述二维矩阵的维度，其它高维描述批量运算
特征值张量	输出	必选	描述返回特征值的张量，特征值不必按照某种顺序进行排列，且w的维度与输入张量mat保持一致
特征向量张量	输出	必选	描述返回特征向量的张量，且其第i列v[:,i]与第i个特征值相对应

7.2.17.10.3 接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

非法参数：表示其他参数出错情况。

7.2.17.11 矩阵范数

7.2.17.11.1 功能

计算张量的范数,这个函数可以计算几个不同的向量范数(1-norm,Euclidean或2-norm,inf-norm,p>0的p-norm）和矩阵范数（Frobenius，1-norm 和 inf-norm）。

7.2.17.11.2 接口参数

矩阵范数函数前向接口应符合表145，C语言示例见A.2.17.11。

表 145 矩阵范数函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示待求范数张量
范数计算类别	输入	可选	表示范数计算的类别，包括向量范数（1-norm、Euclidean或2-norm、inf-norm、p>0的p-norm）和矩阵范数（Frobenius、1-norm 和 inf-norm）。默认值：inf-norm范数
轴数组首地址	输入	可选	如果axis是{}，输入被视为一个向量，整个张量计算出一个范数值。如果axis的数组长度为1，输入被视为batch的向量；如果axis的数组长度是2，则输入视为batch的矩阵
轴数组元素个数	输入	可选	表示轴数组的元素个数。默认值：0
维度保留	输入	可选	如果为true，则输出张量保留输入张量的维度。否则，输出的维度将小于输入的维度默认值：false
输出张量	输出	必选	表示计算结果，包含向量或矩阵的范数。如果keep_dims是true，那么输出张量与输入张量的维度相同。否则，如果axis为{}，则输出为标量；如果axis的数组长度大于1，则输出的张量的维度小于输入张量的维度

7.2.17.11.3 接口返回值

没有错误：表示操作成功。

对象未初始化：表示输入张量对象未初始化。

类型不匹配: 表示输入张量对象的类型和输出张量的类型不匹配。

内存不足：表示输出张量分配空间不足。

非法参数：表示其他参数不合法。

其它内部错误：表示内部的调用操作出错。

7.2.17.12 线性操作

7.2.17.12.1 功能

对输入张量进行线性变换，见式（57）。

$$y = xA^T + b \tag{57}$$

式中：

x——输入张量；

A——权重张量；

b——偏置张量。

7.2.17.12.2 前向接口参数

线性操作函数前向接口应符合表146，C语言示例见A.2.17.12。

表 146 线性操作函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	线性操作输入张量
权重张量	输入	必选	线性操作的权重，形状为[out_features, in_features]的张量
偏置张量	输入	可选	形状为[out_features]的张量，空值表示无偏置。默认值：空值
输出张量	输出	必选	表示计算结果，形状为[N, *, out_features]的张量

7.2.17.12.3 前向接口返回值

没有错误：表示操作成功。

类型不匹配：表示参数的数据类型不一致。

维度不匹配：表示维度不匹配。

7.2.17.12.4 后向接口参数

线性操作函数后向接口应符合表147，C语言示例见A.2.17.12。

表 147 线性操作函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是形状为[N, *, out_features]的张量
权重张量	输入	必选	线性操作的权重，形状为[out_features, in_features]的张量，即前向接口中的权重张量
输入张量梯度	输出	必选	表示前向接口中输入张量的梯度。输入张量是线性操作输入张量

7.2.17.12.5 后向接口返回值

没有错误：表示操作成功。

类型不匹配：表示参数的数据类型不一致。

维度不匹配：表示维度不匹配。

7.2.17.12.6 后向接口参数

线性操作函数后向接口应符合表148，C语言示例见A.2.17.12。

表 148　线性操作函数后向接口函数参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是形状为[N, *, out_features]的张量
输出张量	输入	必选	表示前向接口中输入张量，即线性操作输入张量
权重张量梯度	输入	必选	表示前向接口中权重张量的梯度。权重张量是线性操作的权重，形状为[out_features, in_features]的张量

7.2.17.12.7　后向接口返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.17.13　双线性操作

7.2.17.13.1　功能

对输入张量进行线性变换，见式（58）。

$$y = x_1 A x_2 + b \quad \text{（58）}$$

式中：
x_1——输入张量1；
A ——权重张量；
x_2——输入张量2；
b ——偏置张量。

7.2.17.13.2　前向接口参数

双线性操作函数函数前向接口应符合表149，C语言示例见A.2.17.13。

表 149　双线性操作函数前向接口参数列表

参数名	类型	可选/必选	描述
输入张量1	输入	必选	双线性操作的第一个输入张量，约定其形状为[N, *, in_features1]
输入张量2	输入	必选	双线性操作的第一个输入张量，约定其形状为[N, *, in_features2]
权重张量	输入	必选	双线性操作的权重，形状为[out_features, in_features1, in_features2]的张量。
偏置张量	输入	可选	形状为[out_features]的张量，空值表示无偏置。默认值：空值
输出张量	输出	必选	表示计算结果，形状为[N, *, out_features]的张量

7.2.17.13.3　接口返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.17.13.4　后向接口参数

双线性操作函数后向接口应符合表150，C语言示例见A.2.17.13。

表 150　双线性操作函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是形状为[N, *, out_features]的张量
输入张量2	输入	必选	双线性操作的第一个输入张量，约定其形状为[N, *, in_features2],即前向接口中输入张量2
权重张量	输入	必选	双线性操作的权重，形状为[out_features, in_features1, in_features2]的张量，即前向接口中权重张量
输入张量1梯度	输出	必选	表示前向接口中输入张量1的梯度。输入张量1是双线性操作的第一个输入张量，约定其形状为[N, *, in_features1]

7.2.17.13.5　后向返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.17.13.6　后向接口参数

双线性操作函数后向接口应符合表151，C语言示例见A.2.17.13。

表 151　双线性操作函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是形状为[N, *, out_features]的张量
输入张量1	输入	必选	双线性操作的第一个输入张量，约定其形状为[N, *, in_features1]，即前向接口中的输入张量1
权重张量	输入	必选	双线性操作的权重，形状为[out_features, in_features1, in_features2]的张量，即前向接口中的权重张量
输入张量2梯度	输出	必选	表示前向接口中输入张量2的梯度。输入张量2是双线性操作的第一个输入张量，约定其形状为[N, *, in_features1]

7.2.17.13.7　后向返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.17.13.8　后向接口参数

双线性操作函数后向接口应符合表152，C语言示例见A.2.17.13。

表 152　双线性操作函数后向接口参数列表

参数名	类型	可选/必选	描述
输出张量梯度	输入	必选	表示前向接口中输出张量的梯度。输出张量是形状为[N, *, out_features]的张量
输入张量1	输入	必选	双线性操作的第一个输入张量，约定其形状为[N, *, in_features1]，即前向接口中的输入张量1
输入张量2	输入	必选	双线性操作的第二个输入张量，约定其形状为[N, *, in_features2]，即前向接口中的输入张量2
权重张量梯度	输出	必选	表示前向接口中权重张量的梯度。权重张量是双线性操作的权重，形状为[out_features, in_features1, in_features2]的张量

7.2.17.13.9　后向返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.18　插值操作

7.2.18.1　功能

将输入张量用指定的插值算法进行上/下采样操作，一般多用于图像处理，如果输入代表图像的张量形状为$[N, C, H_{in}, W_{in}]$，其中N是批尺寸、C是通道数、H_{in}是特征高度、W_{in}是特征宽度。若指定输出尺寸为$[H_{out}, W_{out}]$，那么通过指定插值方法计算过后的输出张量形状为$[N, C, H_{out}, W_{out}]$，若指定缩放因子为$[\text{factor}_H, \text{factor}_W]$，那么通过指定插值方法计算过后的输出张量形状为$[N, C, \lfloor H_{in} \times factor_H \rfloor, \lfloor W_{in} \times \text{factor}_W \rfloor]$

7.2.18.2　接口参数

插值函数前向接口应符合表153，C语言示例见A.2.18.1。

表 153　插值函数参数列表

参数名	类型	可选/必选	描述
输入张量	输入	必选	表示需要进行插值的输入张量。包含时间，空间和体积上的采样，即预期输入为三维、四维或五维形状的张量
输出尺寸	输入	可选	插值后张量的输出尺寸，针对三维、四维、五维输入张量，该参数分别指定W维度，[H,W]维度和[D, H, W]的输出尺寸。该参数可为空，表示输出尺寸由缩放因子决定。默认值：空值
缩放因子	输入	必选	插值后张量输出尺寸的缩放倍数，针对三维、四维、五维输入张量，该参数分别指定W维度，[H,W]维度和[D, H, W]输出尺寸的缩放倍数。该参数只有在输出尺寸为空时有效
模式	输入	可选	计算是采用的插值算法，包括最邻近插值（nearest）、线性插值（linear）、双线性插值（bilinear）、三线性插值（trilinear）区域插值（area）等。默认值：最邻近插值（nearest）
输出张量	输出	必选	表示计算结果，形状由输入张量的形状和输出尺寸或缩放因子共同决定

7.2.18.3 接口返回值

没有错误：表示操作成功。
类型不匹配：表示参数的数据类型不一致。
维度不匹配：表示维度不匹配。

7.2.18.4 其他附加说明

输出尺寸和缩放因子不能同时为空。

7.3 算子接口最小集

为适用于深度学习编译器等技术，实现无限阶微分计算，以及降低硬件厂商和人工智能框架厂商适配成本，本文件抽象出了基础的算子集合，形成算子接口最小集，详见表154。人工智能框架应至少具备算子接口最小集中的算子并满足其接口标准。

表 154 算子接口最小集列表

序号	分类	标准章节号	标准定义的接口名称
1	张量创建与销毁	7.2.1.4	按指定值创建稠密张量
2		7.2.1.5	创建未初始化张量
3		7.2.1.7	以均匀分布随机数创建稠密张量
4		7.2.1.8	以正态分布随机数创建稠密张量
5		7.2.1.10	以多项式分布创建稠密张量
6		7.2.1.11	创建随机排列的一维稠密张量
7		7.2.1.13	创建线性空间均匀分布稠密张量
8	张量查询与检查	7.2.2.1	形状查询
9		7.2.2.3	无穷检查
10		7.2.2.4	未定义数检查
11	张量转换	7.2.3.1	转换数据类型
12		7.2.3.2	改变张量形状
13		7.2.3.5	张量转置
14		7.2.3.6	张量分拆
15		7.2.3.7	张量合并
16		7.2.3.11	张量重复
17		7.2.3.12	张量补全
18		7.2.3.14	张量循环滚动变换
19		7.2.3.17	张量聚集
20		7.2.3.21	张量翻转

表 154　算子最小集列表（续）

序号	分类	标准章节号	标准定义的接口名称
21		7.2.3.22	张量正负判断
22		7.2.3.25	张量选择
23	算术操作	7.2.4.1	张量加法操作
24		7.2.4.2	张量减法操作
25		7.2.4.3	张量乘法操作
26		7.2.4.5	张量除法操作
27		7.2.4.6	张量整除操作
28		7.2.4.8	张量取模操作
29		7.2.4.9	张量逐元素取最大值
30		7.2.4.10	张量逐元素取最小值
31	比较操作	7.2.5.1	判断张量是否相等
32		7.2.5.2	判断张量是否不等
33		7.2.5.3	判断张量是否大于
34		7.2.5.4	判断张量是否大于等于
35		7.2.5.5	判断张量是否小于
36		7.2.5.6	判断张量是否小于等于
37	逻辑操作	7.2.6.1	张量的“逻辑与”操作
38		7.2.6.2	张量的“逻辑或”操作
39		7.2.6.3	张量的“逻辑非”操作
40		7.2.6.4	张量的“逻辑异或”操作
41	位操作	7.2.7.1	逐位“与”操作
42		7.2.7.2	逐位“或”操作
43		7.2.7.3	逐位“异或”操作
44		7.2.7.4	逐位反转操作
45	幂操作	7.2.8.1	幂
46	舍入操作	7.2.9.1	向下取整
47		7.2.9.2	向上取整
48		7.2.9.3	截断取整
49		7.2.9.4	就近取整
50	三角函数	7.2.10.1	正弦函数
51		7.2.10.2	余弦函数
52		7.2.10.3	正切函数
53		7.2.10.4	反正弦函数
54		7.2.10.5	反余弦函数
55		7.2.10.6	反正切函数
56	双曲函数	7.2.11.1	双曲正弦函数
57		7.2.11.2	双曲余弦函数
58		7.2.11.3	双曲正切函数
59		7.2.11.4	反双曲正弦函数
60		7.2.11.5	反双曲余弦函数
61		7.2.11.6	反双曲正切函数
62	指对函数	7.2.12.1	指数函数
63		7.2.12.2	指数函数扩展

表 154　算子最小集列表（续）

序号	分类	标准章节号	标准定义的接口名称
64	指对函数	7.2.12.3	以 e 为底的对数函数
65		7.2.12.4	以 e 为底的对数函数扩展
66	规约类操作	7.2.13.1	规约
67		7.2.13.2	前缀和
68	索引操作	7.2.14.1	最大索引
69		7.2.14.2	最小索引
70		7.2.14.3	排序索引
71		7.2.14.4	Top K 索引
72		7.2.14.5	非零索引
73	复数操作	7.2.15.1	复数构建
74		7.2.15.2	复数共轭
75		7.2.15.3	获取虚部
76		7.2.15.4	获取实部
77	线性代数操作	7.2.17.1	矩阵乘法

附 录 A
（资料性附录）
基础数学类算子接口 C 语言参考定义示例

A.1 数据结构

```
#include <stddef.h>
#include <stdint.h>
/* The type code options for DataType */
typedef enum {
  kChar = 0U,
  kBool = 1U,
  kInt = 2U,
  kUInt = 3U,
  kReal = 4U,
  kComplex = 5U,
  ...
} TypeCode;

/* The data type of the elements in the tensor */
typedef struct {
  /* The type code */
  TypeCode code;
  /* Number of bits, common choices are 1,2, 4, 8, 16, 32 etc. */
  uint8_t size;
} DataType;

/* The device type options for Device */
typedef enum {
  kCPU = 0,
  kGPU = 1,
  ...
} DeviceType;

/* The device of the tensor in memory */
typedef struct {
  /* The device type */
  DeviceType type;
  /* The device-id among this specific device type */
  int id;
} Device;
```

```
/* The layout format type of the tensor in memory */
typedef enum {
  /* Dense tensor */
  kDense,
  /* CSR format for sparse tensor */
  kCSR,
  ...
} LayoutType;

/* The layout information of the tensor in memory */
typedef struct {
  /* Layout type */
  LayoutType type;
  /* The layout description of the storage type */
  int64_t* min2maj;
} Layout;

/* The Shape of the tensor in memory */
typedef struct {
  /* Number of dimensions */
  int64_t ndim;
  /* The dimensions of the tensor */
  int64_t* dims;
  /* Layout informations of tensor*/
  Layout layout;
} Shape;

/* The implementation of the storage hold by the tensor */
struct {
/* The data type of the elements in the storage */
  DataType dtype;
  /* The device where the elements are stored */
  Device device;
  /* The number of the elements hold by the storage */
  int64_t size;
  /* The raw data pointer pointing to the memory space */
  void *data;
} StorageImpl;

/* The handle of StorageImpl */
```

```
typedef struct StorageImpl* Storage;

/** The implementation for Tensor */
struct TensorImpl {
  /* The total number of the elements in the tensor */
  int64_t size;
  /* the offset of elements to the beginning pointer to data */
  int64_t offset;
  /* The shape of the tensor */
  Shape shape;
  /* The scale and zero_point can be used to convert the 8/16 bit integer to the real
value */
  /* The formula is: real_value = (integer_value  -  zero_point) * scale */
  float *scale;
  int32_t zero_point;
  /* The Storage of the tensor. */
  Storage storage;
};

/* The handle of TensorImpl */
typedef struct TensorImpl* Tensor;
```

A.2　基础数学操作算子接口

A.2.1　张量创建与销毁

A.2.1.1　拷贝已有数据创建稠密张量

C 语言：

```
Status op_create_tensor(const DataType dtype,
                        const Device device,
                        const Shape shape,
                        const void *values,
                        const int64_t n_bytes,
                        Tensor *output);
```

参数：

dtype (IN)：张量的数据类型。

device(IN)：张量的设备类型。

shape(IN)：张量的形状。

values (IN)：初始化张量数据的数组。

n_bytes(IN)：初始化张量数组的长度。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH ：类型不匹配。STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

STATUS_DIMENSIONS_MISMATCH：表示张量和初始化数组维度不匹配。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
/* data: [[1, 2, 3],[4, 5, 6]] */
Tensor output;
op_create_tensor(FLOAT32, CPU, shape, data, 6 * sizeof(float), &output);
/* output: [[1, 2, 3],[4, 5, 6]] */
```

A.2.1.2 引用已有数据创建稠密张量

C 语言：

```
Status op_wrap_tensor(const DataType dtype,
                      const Device device,
                      const Shape shape,
                      const Tensor values,
                      Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型。

device(IN)：张量的设备类型。

shape(IN)：张量的形状。

values (IN)：初始化张量。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
 /* shape: [2, 3] */
/* data: [[1, 2, 3],[4, 5, 6]] */
Tensor output;
op_wrap_tensor(FLOAT32, CPU, shape, data, &output);
/* output: [[1, 2, 3],[4, 5, 6]] */
```

A.2.1.3 创建全零稠密张量

C 语言：

```
Status op_zeros(const DataType dtype,
                const Device device,
                const Shape shape,
                Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型。

device(IN)：张量的设备类型。

shape(IN)：张量的形状。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
Tensor output;
op_zeros(FLOAT32, CPU, shape, &output);
/* output:
*  [[0, 0, 0],
 *  [0, 0, 0]]
 */
```

A.2.1.4 按指定值创建稠密张量

C 语言：

```
Status op_full(const DataType dtype,
               const Device device,
               const Shape shape,
               const void *value,
               Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型。

device(IN)：张量的设备类型。

shape(IN)：张量的形状。

value (IN)：用来初始化的数据。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
Tensor output;
float value = 1.0
op_full(FLOAT32, CPU, shape, &value, &output);
/* output:
*  [[1.0, 1.0, 1.0],
*   [1.0, 1.0, 1.0]]
 */
```

A.2.1.1 创建未初始化张量

C 语言：

```
Status op_empty(const DataType dtype,
                const Device device,
                const Shape shape,
                Tensor *output);
```

参数：

dtype (IN)：创建张量的数据类型。

device(IN)：张量的设备类型。

shape(IN)：张量的形状。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
Tensor output;
op_empty(FLOAT32, CPU, shape, &output);
/* output:
*  [[0.1, 0.5, 0.9],
```

```
*  [076, 0.3, 0.2]]
*/
```

A.2.1.2 创建连续内存张量

C 语言：

```
Status op_contiguous(const Tensor input,
                     Tensor *output);
```

参数：

input (IN)：被复制的张量。

output (OUT)：输出复制后拥有连续内存的张量。

返回值：

STATUS_SUCCESS: 表示生成连续内存的张量成功。

STATUS_UNINITIALIZED_OBJECT：对象未初始化

STATUS_INVALID_ARGRUMENT: 表示参数出错。

STATUS_ALLOC_FAILED: 表示分配空间不足。

STATUS_INTERNAL_ERROR: 表示内部调用操作出错。

示例：

```
/* input: [1,2,3] */
op_ contiguous (input, output);
/* output: [1,2,3] */
```

A.2.1.3 以均匀分布随机数创建稠密张量

C 语言：

```
Status op_rand_uniform(const DataType dtype,
                       const Device device,
                       const Shape shape,
                       const void *min,
                       const void *max,
                       const int64_t seed,
                       Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型，可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等，默认为 FLOAT32。

device(IN): 张量的设备类型，可以为 CPU，GPU 等，默认为 CPU。

shape(IN): 张量的形状，包括张量的维度，维度大小数组，以及布局信息。其中布局信息没有指定时采用默认布局。

min (IN): 生成随机数所遵循均匀分布最小值，要求与 dtype 所对应数据类型兼容。

max (IN): 生成随机数所遵循均匀分布最大值，要求与 dtype 所对应数据类型兼容。

seed(IN)： 生成随机数的种子。若 seed 为 0，表示使用系统的随机种子；否则，使用 seed 为种子来生成随机数。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
Tensor output;
float min = 0.0;
float max = 2.0;
int64_t seed = 0;
op_rand_uniform(FLOAT32, CPU, shape, &min, &max, seed, &output);
/* output:
* [[1.5104, 0.6955], [0.4895, 0.9185]]
*/
```

A.2.1.4 以正态分布随机数创建稠密张量

C 语言：

```
Status op_rand_normal(const DataType dtype,
                      const Device device,
                      const Shape shape,
                      const void *mean,
                      const void *std,
                      const int64_t seed,
                      Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型，可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

device(IN)：张量的设备类型，可以为 CPU，GPU 等。

shape(IN)：张量的形状，包括张量的维度，维度大小数组，以及布局信息。

mean (IN)：生成随机数所遵循正态分布均值，要求与 dtype 所对应数据类型兼容。

std (IN)：生成随机数所遵循正态分布标准差，要求与 dtype 所对应数据类型兼容。

seed(IN)： 生成随机数的种子。若 seed 为 0，表示使用系统的随机种子；否则，使用 seed 为种子来生成随机数。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default}  */
Tensor output;
float mean = 1.0;
float std = 1.0;
int64_t seed = 0;
op_rand_normal(FLOAT32, CPU, shape, &mean, &std, seed, &output);
/* output:
* [[1.5104, 0.6955], [1.4895, 0.9185]]
*/
```

A.2.1.5 以伯努利分布随机数创建稠密张量

C 语言：

```
Status op_rand_bernoulli (const Tensor input,
                          Tensor *output)
```

参数：

input (IN)：输入的概率值。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* input: [0.1, 0.8, 0.1]  */
Tensor output;
op_rand_bernoulli (input, &output);
/* output:
* [0.,1.,0.]
*/
```

A.2.1.6 以多项式分布创建稠密张量

C 语言：

```
Status op_ multinomial(const Tensor input,
                       const int64_t num_samples,
                       const bool replacement,
                       Tensor *output)
```

参数：

input(IN)：概率张量。

num_samples (IN)：采样次数。

replacement (IN)：是否为有放回采样。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: [[0.56911194, 0.66988271, 0.99126470, 0.31639785]]*/
int64 num_samples = 5;
bool replacement = true;
op_ multinomial (input, num_samples, replacement, &output);
/* output:
* [[0, 2, 2, 2, 3]]
 */
```

A.2.1.7 创建随机排列的一维稠密张量

C 语言：

```
Status op_ randperm(const DataType dtype,
                    const Device device,
                    const int len,
                    Tensor *output)
```

参数：

dtype(IN)：创建张量的数据类型。

device (IN)：创建张量的设备类型。

len (IN)：创建张量的长度，也即可表示范围的最大值，。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: [[0.56911194, 0.66988271, 0.99126470, 0.31639785]]*/
Tensor output;
op_rand_normal(FLOAT32, CPU, 4, &output);
/* output:
 *[[0, 2, 2, 2, 3]]
 */
```

A.2.1.8 以数字序列创建稠密张量

C 语言：

```
Status op_range(const DataType dtype,
                const Device device,
                const void *start,
                const void *limit,
                const void *step,
                Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型，可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

device(IN)：张量的设备类型，可以为 CPU，GPU 等。

start (IN)：创建数字序列的第一个值，要求与 dtype 所对应数据类型兼容。

limit (IN)：创建数字序列的上限，要求与 dtype 所对应数据类型兼容。

step (IN)：生成的相邻两个随机数的变化步长，要求与 dtype 所对应数据类型兼容。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
Tensor output;
float start = 5;
float limit = 3;
float step = -0.5;
op_range(FLOAT32, CPU, &start, &limit, &step, &output);
/* output:
* [5, 4.5, 4, 3.5,3]
```

```
*/
```

A.2.1.9 创建线性空间均匀分布稠密张量

C 语言：

```
Status op_linspace(const DataType dtype,
                   const Device device,
                   const void *start,
                   const void *stop,
                   const int64_t num,
                   Tensor *output)
```

参数：

dtype (IN)：创建张量的数据类型，可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

device(IN)：张量的设备类型，可以为 CPU，GPU 等。

start (IN)：区间的起点，包含在区间内。要求与 dtype 所对应数据类型兼容。

stop (IN)：区间的终点，包含在区间内。要求与 dtype 所对应数据类型兼容。

num (IN)：创建张量的元素个数。

output (OUT)：新创建的张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

示例：

```
Tensor output;
float start = 0;
float stop = 10;
int64_t num = 5;
op_range(FLOAT32, CPU, &start, &stop, num, &output);
/* output:
* [0.0, 2.5, 5.0, 7.5,10.0]
*/
```

A.2.1.10 创建稀疏张量

C 语言：

```
Status op_create_sparse_tensor(const DataType dtype,
                               const Device device,
                               const Shape shape,
                               const int *indices,
```

```
                                const void *values,
                                const int64_t nnzs,
                                Tensor *output)
```

参数：

dtype (IN)：创建稀疏张量的数据类型，可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

device(IN)：张量的设备类型，可以为 CPU，GPU 等。

shape(IN)：张量的形状，包括张量的维度，维度大小数组，以及布局信息。其中布局信息没有指定时采用默认布局。

indice (IN)：创建稀疏张量中非零元的坐标，是二维（nnzs，ndim）数组。

values (IN)：非零元素值组成的一维数组，长度为 nnzs。

nnzs (IN)：非零元的个数。

output (OUT)：新创建的稀疏张量。

返回值：

STATUS_SUCCESS：表示成功创建稀疏张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{kCOO} */
    /* input: indices = [[1,3],[2,4]] */
    /* input: values = [5, 7] */
    Tensor output;
    op_create_sparse_tensor(FLOAT32, CPU, shape, indices, values, 2, &output);
```

A.2.1.11 创建量化张量

C 语言：

```
Status op_create_quant_tensor(const DataType dtype,
                              const Device device,
                              const Shape shape,
                              const void *values,
                              const int64_t n_bytes,
                              const float* scale,
                              const int32_t zero_point,
                              Tensor *output);
```

参数：

dtype (IN)：张量的数据类型。

device(IN)：张量的设备类型，可以为 CPU，GPU 等。调用者需保证初始化数组的设备类型和张量的设备类型保持一致。

shape(IN)：张量的形状，包括张量的维度，维度大小数组，以及布局信息。其中布局信息没有指定时采用默认布局。

values (IN)：初始化量化张量的量化数据数组。

n_bytes(IN)：初始化张量数组的长度。

scale(IN)：scale 为大于零的浮点类型数组。

zero_point(IN)：如果 DataType 为采用对称方式的量化数据类型.

output (OUT)：新创建带有量化信息 scale 和 zero_point 的量化张量。

返回值：

STATUS_SUCCESS：表示成功创建张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

STATUS_DIMENSIONS_MISMATCH：表示张量和初始化数组维度不匹配。

示例：

```
/* shape: {ndim:2, dims:[2, 3], layout{default} */
/* data: [[1, 2, 3],[4, 5, 6]] */
/* scale: [0.25] */
/* zero_point: 0 */
Tensor output;
op_create_quant_tensor(QT_SYM_INT8, CPU, shape, data, 6, scale, zero_point, &output);
/* output: [[1, 2, 3],[4, 5, 6]] with scale[0.25] and zero_point=0 */
```

A.2.1.12 复制张量

C 语言：

```
Status op_duplicate( const Tensor input,
                     Tensor *output);
```

参数：

input (IN)：被复制的张量。

output (OUT)：输出复制操作后的张量。

返回值：

STATUS_SUCCESS：表示复制成功。

STATUS_INVALID_ARGRUMENT：表示参数出错。

STATUS_ALLOC_FAILED：表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部调用操作出错。

示例：

```
/* input: [1,2,3] */
op_duplicate(input, output);
/* output: [1,2,3] */
```

A.2.1.13 复制对角线元素

C 语言：

```
Status op_diag(const Tensor input,
               Tensor *output);
```

参数：

input (IN): 表示 1-D 输入张量，形状为[N]。

output (OUT): 表示 2-D 结果张量，形状为[N, N]。

返回值：

STATUS_SUCCESS: 表示复制成功。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

STATUS_ALLOC_FAILED: 表示创建张量分配空间不足。

STATUS_INTERNAL_ERROR: 表示内部调用操作出错。

示例：

```
/* input: [1,2,3] */
op_diag(input, &output);
/* output: [[1, 0, 0], [0, 2, 0], [0, 0, 3]] */
```

A.2.1.14 销毁张量

C 语言：

```
Status op_destroy(Tensor *input);
```

参数：

input (IN): 待销毁的张量指针。

返回值：

STATUS_SUCCESS: 表示成功销毁张量。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

STATUS_UNINITIALIZED_OBJECT: 表示传入的为空指针，无法销毁。

STATUS_INTERNAL_ERROR: 表示内部调用操作出错。

示例：

```
op_destroy(&input);
```

A.2.2 张量查询与检查

A.2.2.1 形状查询

C语言：

```
Status op_shape(const Tensor input,
```

```
                    Tensor *output);
```

参数：

input (IN)：输入多维张量。

output (OUT)：表示输入张量形状的 1-D 张量，数据类型为 INT32。

返回值：

STATUS_SUCCESS：表示成功查询张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

示例：

```
/* input: [[1,2,3], [4, 5, 6]] */
op_shape(input, &output);
/* output: [2, 3] */
```

A.2.2.2 有限检查

C 语言：

```
Status op_is_finite(const Tensor input,
                    Tensor *output);
```

参数：

input (IN)：输入多维张量。

output (OUT)：表示检查的结果张量，形状为与输入张量形状一致，数据类型为 BOOL。

返回值：

STATUS_SUCCESS：表示成功检查张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

示例：

```
/* input: [[1,2, inf], [4, nan, 6]] */
op_isfinite(input, &output);
/* output: [[false, false, true],[false, true, false]] */
```

A.2.2.3 无穷检查

C 语言:

```
Status op_is_nan(const Tensor input,
                 Tensor *output);
```

参数:

input (IN)：输入多维张量。

output (OUT)：表示检查的结果张量，形状与输入张量形状一致，数据类型为 BOOL。

返回值：

STATUS_SUCCESS：表示成功检查张量。

STATUS_INVALID_ARGRUMENT：表示参数出错。

示例：

```
/* input: [[1,2, nan], [4, 5, 6]] */
op_has_nan(input, &output);
/* output: [[false, false, true],[false, false, false]] */
```

A.2.2.4 未定义数检查

C 语言：

```
Status op_is_inf(const Tensor input,
                 Tensor *output);
```

参数：

input (IN): 输入多维张量。

output (OUT): 表示检查的结果张量，形状与输入张量形状一致，数据类型为 BOOL。

返回值：

STATUS_SUCCESS: 表示成功检查张量。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

示例：

```
/* input: [[1,2, inf], [4, 5, 6]] */
op_has_inf(input, &output);
/* output: [[false, false, true],[false, false,false]] */
```

A.2.2.5 元素个数查询

C 语言：

```
Status op_size(const Tensor input, int& size);
```

参数：

input (IN): 输入张量。

size (OUT): 张量中元素个数。

返回值：

STATUS_SUCCESS: 表示成功查询张量。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

示例：

```
/* input: [4, 5, 6] */
int size;
op_size(input, &size);
/* size: 3 */
```

A.2.2.6 秩查询

C 语言：

```
Status op_rank(const Tensor input, int& rank);
```

参数：

input (IN): 输入张量。

rank (OUT): 张量的秩。

返回值：

STATUS_SUCCESS: 表示成功查询张量。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

A.2.2.7 连续内存检查

C 语言：

```
bool  out;
Status op_ is_contiguous (const Tensor input, &out);
```

参数：

input (IN): 输入多维张量。

out(OUT):表示张量是否连续

返回值：

STATUS_SUCCESS: 表示成功检查张量。

STATUS_UNINITIALIZED_OBJECT：对象未初始化。

STATUS_INVALID_ARGRUMENT: 表示参数出错。

STATUS_INTERNAL_ERROR:其它内部错误。

示例：

```
/* input: [[1,2, 3], [4, 5, 6]] */
bool result;
op_ is_contiguous (input, &result);
/* output: true */
```

A.2.3 张量转换

A.2.3.1 转换数据类型

C 语言：

```
Status op_cast( const Tensor input,
                const DataType type,
                Tensor *output)
```

参数：

input (IN)：表示输入张量。

type (IN): 表示要转换的类型，包括 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

output (OUT)：表示类型转换后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和要转换的类型不兼容。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: data = [1.1, 2.2], type = FLOAT32 */
op_cast(input, INT32, &output);
/* input: data = [1, 2], type = INT32 */
```

A.2.3.2 改变张量形状

C 语言：

```
Status op_reshape( const Tensor input,
                   const int64_t *dims,
                   const int64_t ndim,
                   Tensor *output);
```

参数：

input (IN)：表示输入张量。

dims (IN)：数组，表示新形状每个维度对应的大小。

ndim (IN)：表示 dims 数组的长度。

output (OUT)：表示改变形状后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_DIMENSIONS_MISMATCH：表示输入张量对象的维度总大小和输出张量维度总大小不一致。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: data = [1, 2, 3, 4, 5, 6, 7, 8], shape = [8] */
/* dims = [2, 4], ndim = 2 */
op_reshape(input, dims, ndim, &output);
/* output: data = [[1, 2, 3, 4], [5, 6, 7, 8]], */
/* shape = [2, 4] */
```

A.2.3.3 扩展维度

C 语言：

```
Status op_expand_dims(const Tensor input,
```

```
                                const int axis,
                                Tensor *output);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示插入新维度的位置。

output (OUT)：表示插入新维度后的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量的维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: shape = [2, 3, 4] */
op_expand_dims(input, 0, &output);
/* output: shape = [1, 2, 3, 4] */
op_expand_dims(input, 1, &output);
/* output: shape = [2, 1, 3, 4] */
```

A.2.3.4 删除维度

C 语言：

```
Status op_squeeze( const Tensor input,
                   const int *axis,
                   const int num_axis,
                   Tensor *output);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示要删除维度的位置数组。

num_axis：表示 axis 数组长度。如果是 0，则处理所有维度。

output (OUT)：表示删除维度为 1 的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 含有超出输入张量维度的元素。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: shape = [1, 2, 3, 1, 4, 1, 1] */
op_squeeze(input, NULL, 0, &output);
/* output: shape = [2, 3, 4] */
/* axis = [3, 5] */
op_squeeze(input, axis, 2, &output);
/* output: shape = [1, 2, 3, 4, 1] */
```

A.2.3.5 张量转置

C 语言：

```
Status op_transpose(const Tensor input,
                    const int *prem,
                    const int prem_len,
                    Tensor *output)
```

参数：

input (IN)：表示输入张量。

Prem (IN)：表示置换维度数组。

Prem_len (IN)：表示 prem 维度数组长度。

Output (OUT)：表示转换后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和要转换的类型不兼容。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input:
*   [[1, 2, 3],
*   [4, 5, 6]]
*/
/* prem = [] */
Op_transpose(input, perm, 0, &output);
/* output:
/*  [[1, 4],
*   [2, 5],
*   [3, 6]]
```

```
*/
/* prem = [1, 0]     */
op_transpose(input, perm, 2, &output);
/* same output:
*  [[1, 4],
*   [2, 5],
*   [3, 6]]
*/
```

A.2.3.6 张量分拆

C 语言：

```
Status op_split(const Tensor input,
                const int *split_sizes,
                const int nsplit,
                const int axis,
                Tensor *output_lists);
```

参数：

input (IN)：表示输入张量。

split_sizes (IN)：表示分拆大小数组。如果不为 NULL，则 split_sizes[i]>0 表示第 i 个分拆大小，并且 split_sizes 元素总和必须等于输入张量形状总大小；如果为 NULL，表示每个分拆大小相等，分拆数量由参数 nsplit 确定。

nsplit (IN)：表示分拆数量，即和 split_sizes 长度相等。

axis (IN)：表示要分拆的维度，从 0 开始计数。

output_lists (OUT)：表示分拆后所输出张量数组，其长度为 nsplit。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: shape = [4, 20] */
/* split_sizes: [2, 8, 4, 6] */
op_split(input, split_sizes, 4, 1, output_lists);
/* output_list[0]: shape = [4, 2] */
/* output_list[1]: shape = [4, 8] */
/* output_list[2]: shape = [4, 4] */
```

```
/* output_list[3]: shape = [4, 6] */
```

A.2.3.7 张量合并

C 语言：

```
Status op_concat( const Tensor *input_lists,
                  const int len,
                  const int axis,
                  Tensor *output);
```

参数：

input_lists (IN)：表示输入的张量数组。

len (IN)：表示输入张量数组长度。

axis (IN)：表示要合并的维度。

output (OUT)：表示合并后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input_list[0]: [[1, 1, 1], [2, 2, 2]] */
/* input_list[1]: [[3, 3, 3], [4, 4, 4]] */
op_concat(input_lists, 2 , 0, &output);
/* output: [[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]] */
op_concat(input_lists, 2 , 1, &output);
/* output: [[1, 1, 1, 2, 2, 2], [3, 3, 3, 4, 4, 4]] */
```

A.2.3.8 张量堆叠

C 语言：

```
Status op_stack( const Tensor *input_lists,
                 const int len,
                 const int axis,
                 Tensor *output);
```

参数：

input_lists (IN)：表示需堆叠的输入张量数组。

Len (IN)：表示输入张量数组长度。

Axis (IN)：表示指定堆叠的维度。

Output (OUT)：表示堆叠后的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input_list[0]: [1, 1, 1] */
/* input_list[1]: [2, 2, 2] */
Op_stack(input_lists, 2 , 0, &output);
/* output: [[1, 1, 1], [2, 2, 2]] */
op_stack(input_lists, 2 , 1, &output);
/* output: [[1, 2], [1, 2], [1, 2]] */
```

A.2.3.9 张量拆堆

C 语言：

```
Status op_unstack(const Tensor input,
                  const int axis,
                  const int num,
                  Tensor **output_lists);
```

参数：

input (IN)：表示需拆堆的输入张量。

axis (IN)：表示指定拆堆的维度，从 0 开始计数。

num (IN)：表示输出张量数组的长度，值必须和张量 input 在 axis 维度上的值相同。

output_lists (OUT)：表示拆叠后的输出张量数组，其长度为 num。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: shape = [2, 4, 20] */
op_split(input, 1, 4, &output_lists);
/* output_list[0]: shape = [2, 20] */
```

```
/* output_list[1]: shape = [2, 20] */
/* output_list[2]: shape = [2, 20] */
/* output_list[3]: shape = [2, 20] */
```

A.2.3.10 张量切片

C 语言：

```
Status op_slice(const Tensor input,
                const int *begin,
                const int *size,
                const int *step,
                Tensor *output);
```

参数：

input (IN)：表示输入张量。

begin (IN)：表示每个维度提取起始位置数组，长度为输入张量的维度。

size (IN)：表示每个维度提取大小数组，长度为输入张量的维度。

step (IN)：表示每个维度上提取步长数组，长度为输入张量的维度。

output (OUT)：表示提取后的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 begin 中元素或者 size 中元素或者 step 中元素超出输入张量维度大小。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

后向接口 C 语言：

```
Status op_slice_backward(const Tensor grad_out,
                const Tensor input,
                const int *begin,
                const int *size,
                const int *step,
                Tensor *grad_in);
```

后向接口参数：

grad_out (IN)：表示输出张量的梯度张量。

Input(IN):表示输入张量。

begin (IN)：表示每个维度提取起始位置数组，长度为输入张量的维度。

size (IN)：表示每个维度提取大小数组，长度为输入张量的维度。

step (IN)：表示每个维度上提取步长数组，长度为输入张量的维度。

grad_in (OUT)：表示输入张量的梯度张量。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 begin 中元素或者 size 中元素或者 step 中元素超出输入张量维度大小。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input:
 *   [[1, 2, 3, 4],
 *    [5, 6, 7, 8],
 *    [9, 10, 11, 12],
 *    [13, 14,,15,16]]
 */

/* begin: [0, 1]
 * size:  [2, 3]
 * step:  [1, 1]
 */
op_slice(input, begin, size, step, &output);
/* output:
 *   [[2, 3, 4],
 *    [6, 7, 8]]
 */

/* begin: [0, 1]
 * size:  [2, 3]
 * step:  [2, 2]
 */
op_slice(input, begin, size, step, &output);
/* output:
 *   [[2, 4],
 *    [10, 12]]
 */
```

A.2.3.11 张量重复

C 语言：

```
Status op_tile(const Tensor input,
               const int *repeats,
               Tensor *output);
```

参数：

input (IN)：表示需输入张量数组。

repeats (IN)：表示每个维度重复次数，长度等于输入张量维度。

output (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input:
 *   [[1, 2],
 *    [3, 4]]
 * repeats:
 *   [2, 3]
 */
op_tile(input, repeats, &output);
/* output:
 *   [[1, 2, 1, 2, 1, 2],
 *    [3, 4, 3, 4, 3, 4],
 *    [1, 2, 1, 2, 1, 2],
 *    [3, 4, 3, 4, 3, 4]]
 */
```

A.2.3.12 张量补全

C 语言：

```
Status op_pad( const Tensor input,
               const int *pad_width,
               const int pad_value,
               const PadMode pad_mode,
              Tensor *output);
```

参数：

input (IN)：表示输入张量。

pad_width (IN)：如果输入张量维度为 n，那么 pad_width 数组长度为 2*n。对于维度 i 来说，pad_width[2*i]和 pad_width[2*i + 1]分别表示在第 i 维度数据之前补全的宽度和之后补全的宽度。

pad_value (IN)：表示补全的数值。

pad_mode (IN)：表示补全模式，枚举类型。如果是 PAD_CONST，就采用 pad_value 补全；如果是 PAD_PERIOD 模式，就采用张量本身数据周期补全；如果是 PAD_MIRROR 模式，就采用张量本身数据镜像补全。

output (OUT)：表示补全后的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: [[1, 2, 3], [4, 5, 6]]
 * pad_width: [[1, 1], [2, 2]]
 */
op_pad(input, pad_width, 0, PAD_CONST, &output);
/* output:
 *   [[0, 0, 0, 0, 0, 0, 0],
 *    [0, 0, 1, 2, 3, 0, 0],
 *    [0, 0, 4, 5, 6, 0, 0],
 *    [0, 0, 0, 0, 0, 0, 0]]
 */

op_pad(input, pad_width, 0, PAD_PERIOD, &output);
/* output:
 *   [[6, 5, 4, 5, 6, 5, 4],
 *    [3, 2, 1, 2, 3, 2, 1],
 *    [6, 5, 4, 5, 6, 5, 4],
 *    [3, 2, 1, 2, 3, 2, 1]]
 */

op_pad(input, pad_width, 0, PAD_MIRROR, &output);
/* output:
```

```
*   [[2, 1, 1, 2, 3, 3, 2],
*    [2, 1, 1, 2, 3, 3, 2],
*    [5, 4, 4, 5, 6, 6, 5],
*    [5, 4, 4, 5, 6, 6, 5]]
*/
```

A.2.3.13 张量逆序变换

C 语言：

```
Status op_reverse(const Tensor input,
                  const int *axis,
                  const int axis_len,
                  Tensor *output);
```

参数：

input (IN)：表示输入张量。

axis (IN)：指定逆序运算的轴的数组，从 0 开始计数。

axis_len (IN)：表示 axis 参数数组的长度。

output (OUT)：表示输出张量，形状、数据类型与张量 input 相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: shape = [4, 3, 2]
/* input: data = [[[ 0,  1], [ 2,  3], [ 4,  5]],
                  [[ 6,  7], [ 8,  9], [10, 11]],
                  [[12, 13], [14, 15], [16, 17]],
                  [[18, 19], [20, 21], [22, 23]]] */
/* axis = [0, 2] */
op_reverse(input, axis, 2, &output);
/* output: data = [[[19, 18], [21, 20], [23, 22]],
                   [[13, 12], [15, 14], [17, 16]],
                   [[ 7,  6], [ 9,  8], [11, 10]],
                   [[ 1,  0], [ 3,  2], [ 5,  4]]] */
```

A.2.3.14 张量循环滚动变换

C 语言：

```
Status op_roll(const Tensor input,
               const int shifts_len,
               const int64_t *shifts,
               const int64_t *axis,
               Tensor *output);
```

参数：

input (IN)：表示输入张量，每个元素的数据类型可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

shifts_len (IN)：表示 shifts 参数的长度。

shifts (IN)：表示每个维度上的滚动位移，数组长度由 shifts_len 参数指定。

axis (IN)：表示滚动的轴，可以指定多个维度。可以为 NULL，此时输入张量会按照 1-D 张量执行滚动变换。若不为 NULL，长度必须为 shifts_len。

output (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: shape = [3, 3]
/* input: data = [[1.0, 2.0, 3.0],
                  [4.0, 5.0, 6.0],
                  [7.0, 8.0, 9.0]] */
/* shifts = [1] */
/* axis = [0] */
op_reverse(input, 1, shifts, axis, &output);
/* output: data = [[7.0, 8.0, 9.0],
                   [1.0, 2.0, 3.0],
                   [4.0, 5.0, 6.0]] */
```

A.2.3.15 张量形状裁剪

C 语言：

```
Status op_crop(const Tensor input,
               const int *shape,
               const int *offsets,
               Tensor *output);
```

参数：

input (IN)：表示输入张量。

shape (IN)：表示输出张量的形状，数组长度与张量 input 的维数相同。

offsets (IN)：表示每个维度长裁剪的偏移量，数组长度与张量 input 的维数相同。

output (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: shape = [3, 5] */
/* input: data = [[0, 1, 2, 0, 0],
                  [0, 3, 4, 0, 0],
                  [0, 0, 0, 0, 0]] */
/* shape = [2, 2] */
/* offsets = [0, 1] */
op_crop(input, shape, offsets, &output);
/* output: data = [[1, 2], [3, 4]] */
```

A.2.3.16 张量数值裁剪

C 语言：

```
Status op_clip(const Tensor input,
               const void *low,
               const void *high,
               Tensor *output);
```

参数：

input (IN)：表示输入张量。

low (IN)：表示区间的下限，张量 input 中小于该值的元素将由该值代替，数据类型与张量 input 在计算上兼容。

high (IN)：表示区间的上限，张量 input 中大于该值的元素将由该值代替，数据类型与张量 input 在计算上兼容。

output (OUT)：表示输出张量，形状、数据类型与张量 input 相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

后向 C 语言：

```
Status op_clip_backward(const Tensor input,
```

```
                    const void *low,
                    const void *high,
                    Tensor *output);
```

后向参数：

grad_in (IN)：表示输出张量的梯度张量。

low (IN)：表示区间的下限，张量 input 中小于该值的元素将由该值代替，数据类型与张量 input 在计算上兼容。

high (IN)：表示区间的上限，张量 input 中大于该值的元素将由该值代替，数据类型与张量 input 在计算上兼容。

grad_out (OUT)：表示输入张量的梯度张量。

后向返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: data = [[0, 1, -2,], [0, -3, 4], [-5, 6, -7]] */
/* low = -1.0 */
/* high = 1.0 */
op_clip(input, &low, &high, &output);
/* output: data = [[1, 2], [3, 4]] */
```

A.2.3.17 张量聚集

C 语言：

```
Status op_gather(const Tensor input,
                 const Tensor index,
                 Tensor *output);
```

参数：

input (IN)：表示输入张量。

index (IN)：表示索引张量，维度必须大于 1 且小于等于张量 input 的维度。

output (OUT)：表示拼接后的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: [[1, 2], [3, 4], [5, 6]] */
/* index: [1, 2] */
op_gather(input, index, &output);
```

```
/* output: [[3, 4], [5, 6]] */
```

A.2.3.18 张量发散更新

C 语言：

```
Status op_scatter(const Tensor input,
                  const Tensor index,
                  const Tensor updates,
                  bool override,
                  Tensor *output);
```

参数：

input (IN)：表示输入张量。

index (IN)：表示索引张量，维度必须大于 1 且小于等于张量 input 的维度。

updates (IN)：表示更新数据张量。

override (IN)：如果张量 index 中的索引值有重复，override=true 时旧更新值将被新更新值覆盖，override=false 时新更新值将与旧更新值相加。

output (OUT)：表示拼接后的输出张量，形状、数据类型与张量 input 相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: [[1, 1], [2, 2], [3, 3]] */
/* index: [2, 1, 1] */
/* updates: [[10, 10], [20, 20], [30, 30]] */
op_scatter(input, index, updates, false, &output);
/* output: [[1, 1], [30, 30], [10, 10]] */
op_scatter(input, index, updates, true, &output);
/* output: [[1, 1], [52, 52], [13, 13]] */
```

A.2.3.19 扩张张量

C 语言：

```
Status op_expand(const Tensor input,
                 const int *shape,
                 Tensor *output);
```

参数：

input (IN)：表示输入张量。

shape (IN)：表示输出张量的形状。

output (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: data = [1, 2, 3] */
/* shape = [2, 3] */
op_expand(input, shape, &output);
/* output: data = [[1, 2, 3], [1, 2, 3]] */
```

A.2.3.20 展平张量

C 语言：

```
Status op_flatten(const Tensor input,
                  const int start_axis,
                  const int stop_axis,
                  Tensor *output);
```

参数：

input (IN)：表示输入张量。

start_axis (IN)：表示需要展平的起始维度。

stop_axis(IN)：表示需要展平的结束维度。

output (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_AXIS_ARGUMENT：表示起始维度或结束维度不合法。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: data = [[[[0., 0.01000000, 0.02000000, 0.03000000],
                   [0.04000000, 0.05000000, 0.06000000, 0.07000000],
                   [0.08000000, 0.09000000, 0.09999999, 0.11000000]],
```

```
                [[0.12000000, 0.13000000, 0.14000000, 0.14999999],
                [0.16000000, 0.17000000, 0.17999999, 0.19000000],
                [0.19999999, 0.20999999, 0.22000000, 0.22999999]]]*/
    /* start_axis = 1 */
    /* stop_axis = 2 */
    op_flatten(input, start_axis, stop_axis, &output);
    /* output: data = [[[0., 0.01000000, 0.02000000, 0.03000000],
                [0.04000000, 0.05000000, 0.06000000, 0.07000000],
                [0.08000000, 0.09000000, 0.09999999, 0.11000000],
                [0.12000000, 0.13000000, 0.14000000, 0.14999999],
                [0.16000000, 0.17000000, 0.17999999, 0.19000000],
                [0.19999999, 0.20999999, 0.22000000, 0.22999999]]] */
```

A.2.3.21 张量翻转

C 语言：

```
Status op_flip(const Tensor input,
               const int *axis,
               Tensor *output);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示需要翻转的轴。

output (OUT)：表示翻转后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_AXIS_ARGUMENT：表示轴不合法。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: data = [[1, 2], [3, 4]] */
/* axis = [1] */
op_flip(input, axis, &output);
/* output: data = [[2, 1], [4, 3]] */
```

A.2.3.22 张量正负判断

C 语言：

```
Status op_sign(const Tensor input,
               Tensor *output);
```

参数：

input (IN)：表示输入张量。

output (OUT)：表示正负判断后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：其它内部错误。

示例：

```
/* input: data = [[1, -2], [-3, 0]] */
op_sign(input, &output);
/* output: data = [[1,-1], [-1, 0]] */
```

A.2.3.23 条件判断组合张量

C 语言：

```
Status op_where(const Tensor condition,
                const Tensor x,
                const Tensor y,
                Tensor *output);
```

参数：

condition (IN)：表示选择 x 或 y 张量中元素的条件。

x (IN)：表示输入张量 x。

y (IN)：表示输入张量 y。

output (OUT)：表示条件判断后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：其它内部错误

示例：

```
/* input: x = [0.9383, 0.1983, 3.2, 1.2] */
/* input: y = [1.0, 1.0, 1.0, 1.0] */
/* input: condition = x > 1 */
op_where(condition, x, y, &output);
/* output: data = [1.0, 1.0, 3.2, 1.2]*/
```

A.2.3.24 一维张量扩充

C 语言：

```
Status op_meshgrid(const Tensor *input,
                   Tensor *output);
```

参数：

input (IN)：表示 k 个一维张量，形状分别为(N1,), (N2,), …, (Nk,)。

output (OUT)：表示扩充后的 k 个形状为(N1, N2, …, Nk)的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_ALLOC_FAILED：内存分配不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: input1= [1, 2] */
/* input: input2= [3, 4, 5] */
op_meshgrid(input1, input2, &output);
/* output: data1 = [[1, 1, 1], [2, 2, 2]] */
/* output: data2 = [[3, 4, 5], [3, 4, 5]] */
```

A.2.3.25 张量选择

C 语言：

```
Status op_ masked_select (const Tensor input,
                          const Tensor mask,
                          Tensor *output);
```

参数：

input (IN)：表示输入张量。

mask (IN)：表示用于索引的二进制掩码的张量。

output (OUT)：表示拼接后的输出张量，数据类型与张量 input 相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: [[1, 2], [3, 4]] */
/* mask: [[true, false], [false, true]] */
op_ masked_select (input, mask, &output);
/* output: [1, 4] */
```

A.2.4 算术操作

A.2.4.1 张量加法操作

C 语言：

```
Status op_add(const Tensor x,
              const Tensor y,
              Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x:  [1.1, 2.2] */
/* y:  [1.1, 2.2] */
op_add(x, y, z);
/* z:  [2.2, 4.4] */
```

A.2.4.2 张量减法操作

C 语言：

```
Status op_sub(const Tensor x,
              const Tensor y,
              Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x:  [1.1, 2.2] */
/* y:  [1., 2.] */
op_sub(x, y, z);
/* z:  [0.1, 0.2] */
```

A.2.4.3 张量乘法操作

C 语言：

```
Status op_mul(const Tensor x,
              const Tensor y,
              Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1.1, 2.2]   */
/* y: [1.0, 2.0]   */
op_mul(x, y, z);
/* z: [1.1, 4.4]   */
```

A.2.4.4 张量乘加操作

C 语言：

```
Status op_muladd(const Tensor x,
                 const Tensor y,
                 const Tensor a,
                 Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

a (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1.1, 2.2]   */
/* y: [1.0, 2.0]   */
/* a: [0.4, 0.1]   */
op_muladd(x, y, a, z);
/* z: [1.5, 4.5]   */
```

A.2.4.5 张量除法操作

C 语言：

```
Status op_div(const Tensor x,
              const Tensor y,
              Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1.1, 2.2]  */
/* y: [1.0, 2.0]  */
op_div(x, y, z);
/* z: [1.1, 1.1]  */
```

A.2.4.6 张量整除操作

C 语言：

```
Status op_floordiv(const Tensor x,
                   const Tensor y,
                   Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [2, 3, 4]   */
/* y: [1, 5, 2]   */
op_floordiv(x, y, &z);
/* z: [2, 0, 2]   */
```

A.2.4.7 张量真除法操作

C 语言：

```
Status op_true_divide(const Tensor x,
                      const Tensor y,
                      Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

示例：

```
/* x: [2, 3, 4]   */
/* y: [1, 5, 2]   */
op_floordiv(x, y, &z);
/* z: [2.0, 0.6, 2.0]   */
```

A.2.4.8 张量取模操作

C 语言：

```
Status op_mod(const Tensor x,
              const Tensor y,
              Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: data = [3, 7], type = INT32*/
/* y: data = [2, 2], type = INT32*/
op_mod(x, y, z);
/* z: data = [1, 1], type = INT32 */
```

A.2.4.9 张量逐元素取最大值

C 语言：

```
Status op_max( const Tensor x,
               const Tensor y,
               Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [2, 3, 4]   */
/* y: [1, 5, 2]   */
op_max(x, y, &z);
/* z: [2, 5, 4]   */
```

A.2.4.10 张量逐元素取最小值

C 语言：

```
Status op_min( const Tensor x,
               const Tensor y,
               Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [2, 3, 4]   */
/* y: [1, 5, 2]   */
op_min(x, y, &z);
/* z: [1, 3, 4]   */
```

A.2.4.11 张量绝对值操作

C 语言：

```
Status op_abs(const Tensor x,
              Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [-1, 1, -2]   */
op_abs(x, y);
/* y: [1, 1, 2]     */
```

A.2.4.12 张量取倒数操作

C 语言：

```
Status op_reciprocal(const Tensor x,
                     Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [-1, 2, -4]          */
op_reciprocal(x, y);
/* y: [1, 0.5, -0.25]      */
```

A.2.4.13 张量对角线元素求和操作

C 语言：

```
Status op_trace(const Tensor x,
                const int64_t offset,
                const int64_t dim0,
                const int64_t dim1,
                Tensor *y);
```

参数：

x (IN)：表示输入张量，每个元素的数据类型可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

offset (IN)：表示对角线的偏移位置，指定二维平面中对角线相对主对角线的偏移。0 代表主对角线，负数代表主对角线左下的对角线，正数代表主对角线右上的对角线。

dim0 (IN)：表示对角线所在二维平面的第一维。

dim1 (IN)：表示对角线所在二维平面的第二维。

y (OUT)：表示输出张量，数据类型与输入张量相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

后向接口 C 语言：

```
Status op_trace_backward(const Tensor grad_y,
                const int64_t offset,
                const int64_t dim0,
                const int64_t dim1,
                Tensor *grad_x);
```

后向接口参数：

grad_y (IN)：表示输出张量的梯度张量。

offset (IN)：表示对角线的偏移位置，指定二维平面中对角线相对主对角线的偏移。0 代表主对角线，负数代表主对角线左下的对角线，正数代表主对角线右上的对角线。

dim0 (IN)：表示对角线所在二维平面的第一维。

dim1 (IN)：表示对角线所在二维平面的第二维。

grad_x (OUT)：表示输入张量的梯度张量。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_INVILID_ARGUMENT：非法参数。

示例：

```
/* x.shape: [2, 3, 3]
   x.data: [[[1, 2, 3], [4, 5, 6]],
            [[1, 2, 3], [4, 5, 6]]] */
op_trace(x, 0, 0, 1, y);
/* y: [5, 7, 9]    */
```

A.2.5 比较操作

A.2.5.1 判断张量是否相等

C 语言：

```
Status op_equal( const Tensor x,
                 const Tensor y,
                 Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1, 2, 3] */
/* y: [1, 3, 3] */
op_equal(x, y, z);
/* z: [true, false, true]   */
```

A.2.5.2 判断张量是否不等

C 语言：

```
Status op_not_equal(const Tensor x,
                    const Tensor y,
                    Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1, 2, 3] */
/* y: [1, 3, 3] */
op_not_equal(x, y, z);
/* z: [false, true, false]   */
```

A.2.5.3 判断张量是否大于

C 语言：

```
Status op_greater( const Tensor x,
                   const Tensor y,
                   Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1, 3, 3] */
/* y: [1, 2, 3] */
op_greater(x, y, z);
/* z: [false, true, false]   */
```

A.2.5.4 判断张量是否大于等于

C 语言：

```
Status op_greater_equal(const Tensor x,
                        const Tensor y,
                        Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1, 3, 3] */
/* y: [1, 2, 3] */
op_greater_equal(x, y, z);
/* z: [true, true, true]    */
```

A.2.5.5 判断张量是否小于

C 语言：

```
Status op_less(const Tensor x,
               const Tensor y,
               Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1, 2, 3] */
/* y: [1, 3, 3] */
op_less(x, y, z);
/* z: [false, true, false]   */
```

A.2.5.6 判断张量是否小于等于

C 语言：

```
Status op_less_equal( const Tensor x,
                      const Tensor y,
                      Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1, 2, 3] */
/* y: [1, 3, 3] */
op_less_equal(x, y, z);
/* z: [true, true, true]   */
```

A.2.5.7 判断张量是否值相近

C 语言：

```
Status op_allclose( const Tensor x,
                    const Tensor y,
                    const double rtol,
                    const double atol,
                    const bool equal_nan,
                    Tensor *z);
```

参数：

x (IN)：表示输入张量，每个元素的数据类型可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64 等。

y (IN)：表示对比张量，数据类型必须与输入张量在计算上兼容。

rtol (IN)：表示相对容忍误差，通常设置为 1e-5。

atol (IN)：表示绝对容忍误差，通常设置为 1e-8。

equal_nan (IN)：表示 NaN 判断标识。如果设置成 true，则两个 NaN 认为是相等的。通常设置为 false。

z (OUT)：表示输出张量，形状为[1]，元素的数据类型为 BOOL。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_INVILID_ARGUMENT：非法参数。

示例：

```
/* x: [10, 1e-7] */
/* y: [10.1, 1e-8] */
op_allclose(x, y, 1e-5, 1e-8, false, z);
/* z: [false] */
```

A.2.6 逻辑操作

A.2.6.1 张量的“逻辑与”操作

C 语言：

```
Status op_logical_and(const Tensor x,
                      const Tensor y,
```

```
                    Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [true, false, true]  */
/* y: [true, true,  true]  */
op_logical_and(x, y, z);
/* z: [true, false, true]  */
```

A.2.6.2 张量的“逻辑或”操作

C 语言：

```
Status op_logical_or(const Tensor x,
                     const Tensor y,
                     Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [true, false, true] */
/* y: [true, true,  true] */
op_logical_or(x, y, z);
/* z: [true, true, true]    */
```

A.2.6.3 张量的“逻辑非”操作

C 语言：

```
Status op_logical_not( const Tensor x,
                       Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [true, false, true]   */
op_logical_not(x, y);
/* y: [false, true, false]    */
```

A.2.6.4 张量的“逻辑异或”操作

C 语言：

```
Status op_logical_xor( const Tensor x,
                       const Tensor y,
                       Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [true, false, true] */
/* y: [true, true,  true] */
op_logical_xor(x, y, z);
/* z: [false, true, false]  */
```

A.2.7 位操作

A.2.7.1 逐位“与”操作

C 语言：

```
Status op_bitwise_and(const Tensor x,
                      const Tensor y,
                      Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1, 1, 1] */
/* y: [2, 0, 1] */
Tensor z;
op_bitwise_and(x, y, &z);
/* z: [0, 0, 1] */
```

A.2.7.2 逐位“或”操作

C 语言：

```
Status op_bitwise_or( const Tensor x,
                      const Tensor y,
                      Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [1, 1, 1] */
/* y: [2, 0, 1] */
Tensor z;
op_bitwise_or(x, y,& z);
/* z: [3, 1, 1] */
```

A.2.7.3 逐位“异或”操作

C 语言：

```
Status op_bitwise_xor( const Tensor x,
                       const Tensor y,
                       Tensor *z);
```

参数：

x (IN)：表示输入张量。

y (IN)：表示输入张量。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1, 1, 1] */
/* y: [2, 0, 1] */
Tensor z;
op_bitwise_xor(x, y, &z);
/* z: [3, 1, 0] */
```

A.2.7.4 逐位反转操作

C 语言：

```
Status op_bitwise_not( const Tensor x,
                       Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [B00001111, B11110000] */
Tensor y;
op_bitwise_invert(x, &y);
/* y: [B11110000, B00001111] */
```

A.2.7.5 逐位左移操作

C 语言：

```
Status op_bitwise_left_shift(const Tensor x,
                             Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [B00000000000000000000000000000001] */
Tensor y;
```

```
op_bitwise_left_shift(x, &y);
/* y: [B00000000000000000000000000000010] */
```

A.2.7.6 逐位右移操作

C 语言：

```
Status op_bitwise_right_shift(const Tensor x,
                              Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [B00000000000000000000000000000001] */
Tensor y;
op_bitwise_right_shift(x, &y);
/* y: [B00000000000000000000000000000000] */
```

A.2.8 幂操作

A.2.8.1 幂

C 语言：

```
Status op_pow( const Tensor x,
               const Tensor e,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

e (IN)：表示输入张量对应元素的幂。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [1, 2, 3] */
/* e: [1, 2, 3] */
Tensor y;
op_pow(x, e, &y);
```

```
/* y: [1, 4, 9]        */
```

A.2.8.2 平方根

C 语言：

```
Status op_sqrt(const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVILID_ARGUMENT：非法参数。

示例：

```
/* x: [4, 9, 16] */
Tensor y;
op_sqrt(x, &y);
/* y: [2, 3, 4]  */
```

A.2.8.3 平方根倒数

C 语言：

```
Status op_rsqrt(const Tensor x,
                Tensor *y)
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVILID_ARGUMENT：非法参数。

示例：

```
/* x: [4, 16, 25] */
Tensor y;
op_rsqrt(x, &y);
/* y: [0.5, 0.25, 0.2]  */
```

A.2.8.4 平方数

C 语言：

```
Status op_square(const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [2, 3, 4]   */
Tensor y;
op_square(x,& y);
/* y: [4, 9, 16]  */
```

A.2.9 舍入操作

A.2.9.1 向下取整

C 语言：

```
Status op_floor(const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。数据类型与 x 一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [3.2, 2, 2.8]   */
Tensor y;
op_floor(x,& y);
/* y: [3, 2, 2]       */
```

A.2.9.2 向上取整

C 语言：

```
Status op_ceil(const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。数据类型与 x 一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [3.2, 2, 2.8]  */
Tensor y;
op_ceil(x, &y);
/* y: [4, 2, 3]      */
```

A.2.9.3 截断取整

C 语言：

```
Status op_trunc(const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。数据类型与 x 一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [-2.1, -1.9, 1.9]   */
Tensor y;
op_trunc(x, &y);
/* y: [-2, -1, 1]         */
```

A.2.9.4 就近取整

C 语言：

```
Status op_round( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。数据类型与 x 一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [3.2, 2, 2.8] */
```

```
Tensor y;
op_round(x,& y);
/* y: [3, 2, 3]      */
```

A.2.9.5 通用舍入取整

C 语言：

```
Status op_rint(const Tensor x,
               const RoundMode mode,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

mode (IN)：表示取整模式，为枚举类型，可以是 DOWNWARD, UPWARD, TOWARDZERO, TONEAREST, 分别对应 op_floor, op_ceil, op_trunc, op_round。

y (OUT)：表示输出张量。数据类型与 x 一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [0.5, -1.5, 2.8]   */
Tensor y;
op_rint(x, TONEAREST,y);
/* y: [0, -2, 2]         */
```

A.2.10 三角函数

A.2.10.1 正弦函数

C 语言：

```
Status op_sin( const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [-π/2,0, π/2]   */
```

```
Tensor y;
op_sin(x, &y);
/* y: [-1, 0, 1]        */
```

A.2.10.2 余弦函数

C 语言：

```
Status op_cos( const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [π,0,π/2]    */
Tensor y;
op_cos(x, &y);
/* y: [-1, 1, 0]        */
```

A.2.10.3 正切函数

C 语言：

```
Status op_tan( const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [π/4,-π/4,0]    */
Tensor y;
```

```
op_tan(x, &y);
/* y: [1, -1, 0]        */
```

A.2.10.4 反正弦函数

C 语言：

```
Status op_asin( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_asin(x, &y);
/* y: [0, 1.57, -1.57]        */
```

A.2.10.5 反余弦函数

C 语言：

```
Status op_acos( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
```

```
op_acos(x, &y);
/* y: [1.57080, 0, 3.14159]   */
```

A.2.10.6 反正切函数

C 语言：

```
Status op_atan( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT: 表示参数不合法。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_atan(x, &y);
/* y: [0, 0.7854, -0.7854]   */
```

A.2.11 双曲函数

A.2.11.1 双曲正弦函数

C 语言：

```
Status op_sinh( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_sinh(x, &y);
/* y: [0, 1.1752, -1.1752]   */
```

A.2.11.2 双曲余弦函数

C 语言：

```
Status op_cosh( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_cosh(x, &y);
/* y: [1, 1.5431, 1.5431]   */
```

A.2.11.3 双曲正切函数

C 语言：

```
Status op_tanh( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_tanh(x, &y);
/* y: [1, 0.76159, -0.76159]   */
```

A.2.11.4 反双曲正弦函数

C 语言：

```
Status op_asinh( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [0,1,-1]   */
Tensor y;
op_asinh(x, &y);
/* y: [0, 0.88137, -0.88137]   */
```

A.2.11.5 反双曲余弦函数

C 语言：

```
Status op_acosh( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT: 表示参数不合法。

示例：

```
/* x: [3,4,5]   */
Tensor y;
op_acosh(x, &y);
/* y: [1.7627,2.0634,2.2924]   */
```

A.2.11.6 反双曲正切函数

C 语言：

```
Status op_atanh( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT：表示参数不合法。

示例：

```
/* x: [-0.5,0,0.5]   */
Tensor y;
op_atanh(x, &y);
/* y: [-0.54931,0,0.54931]       */
```

A.2.12 指对函数

A.2.12.1 指数函数

C 语言：

```
Status op_exp( const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [-0.5,0,0.5]   */
Tensor y;
op_exp(x, &y);
/* y: [0.60653,1,1.64872]   */
```

A.2.12.2 指数函数扩展

C 语言：

```
Status op_expm1( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

示例：

```
/* x: [0,1]   */
Tensor y;
op_expm1(x,& y);
/* y: [0,e-1]   */
```

A.2.12.3 以e为底的对数函数

C 语言：

```
Status op_log( const Tensor x,
               Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT: 非法参数，比如输入张量元素值超出该函数取值范围。

示例：

```
/* x: [1,e]   */
Tensor y;
op_log(x,& y);
/* y: [0,1]   */
```

A.2.12.4 以e为底的对数函数扩展

C 语言：

```
Status op_log1p( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT：表示参数不合法。

示例：

```
/* x: [0.0]   */
Tensor y;
op_log1p(x,& y);
/* y: [0]   */
```

A.2.12.5 以10为底的对数函数

C 语言：

```
Status op_log10( const Tensor x,
                 Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT: 表示参数不合法。

示例：

```
/* x: [1, 10, 100]   */
Tensor y;
op_log10(x,& y);
/* y: [0, 1, 2]       */
```

A.2.12.6 以2为底的对数函数

C 语言：

```
Status op_log2( const Tensor x,
                Tensor *y);
```

参数：

x (IN)：表示输入张量。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_TYPE_MISMATCH: 表示参数的数据类型不一致。

STATUS_INVALID_ARGUMENT: 表示参数不合法。

示例：

```
/* x: [1, 2, 4]   */
Tensor y;
op_log2(x,& y);
/* y: [0, 1, 2]   */
```

A.2.13 规约类操作

A.2.13.1 规约

C 语言：

```
Status op_reduce(const Tensor input,
                 const ReduceMode mode,
                 const int *axis,
                 const int axis_len,
                 const _Bool keep_dims,
                 Tensor *output)
```

参数：

input (IN)：表示需要归约的张量，其元素类型包括 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

mode (IN)：表示规约的类型，其是枚举类型变量，其定义为

```
enum ReduceMode {
  SUM,      //求和规约
  PROD,     // 乘积规约
  MAX,      //求最大规约
  MIN,      //求最小规约
  MEAN,     //求均值规约
  LAND,     // 逻辑与
  LOR,      // 逻辑或
  LXOR,     // 逻辑异或
  BAND,     // 位与
  BOR,      // 位或
  BXOR,     // 位异或
  …
};
```

除了目前列举的规约类型，ReduceMode 以后还应支持自定义的规约类型。

axis (IN)：表示需要规约的维度，其类型为 int 型数组，且其数组长度的范围是[0,rank(input)]内，当其值为{}时，则规约所有维度，当其值为{0}，表示规约 0 维度，当其值为{1}，表示规约 1 维度，当其值为{0, 1}表示既要规约 0 维度也要规约 1 维度。

axis_len (IN)：表示 axis 数组的长度。

keep_dims (IN)：如果为 true，则规约时保留张量的维度；如果为 false 时，按 axis 指定的维度降低一维进行规约。

output (OUT)：表示对输入张量进行规约后的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和输出规约张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

后向接口 C 语言：

```
Status op_reduce_bwd(const Tensor out_grad,
                     const Tensor input,
                     const ReduceMode mode,
                     const int *axis,
                     const int axis_len,
```

```
                    const _Bool keep_dims,
                    Tensor *in_grad)
```

后向接口参数：

out_grad (IN)：表示输出张量的梯度。

input (IN)：表示需要归约的张量，其元素类型包括 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64, COMPLEX64, COMPLEX128 等。

mode (IN)：表示规约的类型，其是枚举类型变量，其定义为

```
enum ReduceMode {
  SUM,      //求和规约
  PROD,     // 乘积规约
  MAX,      //求最大规约
  MIN,      //求最小规约
  MEAN,     //求均值规约
  LAND,     // 逻辑与
  LOR,      // 逻辑或
  LXOR,     // 逻辑异或
  BAND,     // 位与
  BOR,      // 位或
  BXOR,     // 位异或
  …
};
```

除了目前列举的规约类型，ReduceMode 以后还应支持自定义的规约类型。

axis (IN)：表示需要规约的维度，其类型为 int 型数组，且其数组长度的范围是[0,rank(input)]内，当其值为{}时，则规约所有维度，当其值为{0}，表示规约 0 维度，当其值为{1}，表示规约 1 维度，当其值为{0, 1}表示既要规约 0 维度也要规约 1 维度。

axis_len (IN)：表示 axis 数组的长度。

keep_dims (IN)：如果为 true，则规约时保留张量的维度；如果为 false 时，按 axis 指定的维度降低一维进行规约。

in_grad (OUT)：表示对输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出规约张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input = [[1, 1, 1], [1, 1, 1]] */
/* axis = [] */
```

```
Tensor output;
op_reduce(input, SUM, axis, 0, false, &output);
/* output = 6 */
/* axis = [0] */
op_reduce(input, SUM, axis, 1, false, &output);
/* output = [2, 2, 2] */
/* axis = [1] */
op_reduce(input, SUM, axis, 1, false, &output) ;
/* output = [3, 3] */
/* axis = [1] */
op_reduce(input, SUM, axis, 1, true, &output) ;
/* output = [[3], [3]] */
/* axis = [0, 1] */
op_reduce(input, SUM, axis, 2, false, &output) ;
/* output = 6 */
/* input = [[1., 1.], [2., 2.]] */
/* axis = [] */
op_reduce(input, MEAN, axis, 0, false, &output) ;
/* output = 1.5 */
/* axis = [0] */
op_reduce(input, MEAN, axis, 1, false, &output) ;
/* output = [1.5, 1.5] */
/* axis = [1] *
op_reduce(input, MEAN, axis , 1, false, &output) ;
/* output = [1., 2.] */
```

A.2.13.2 前缀和

C 语言：

```
Status op_cumsum(const Tensor input,
                 const int axis,
                 Tensor *output);
```

参数：

input (IN)：表示需要进行累加的输入张量，每个元素的数据类型可以为 UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64 等。

axis (IN)：表示输入张量累加的轴，-1 代表按照 1-D 张量求全局的前缀和。

output (OUT)：表示输出张量，元素数据类型由 dtype 参数指定，或者与输入张量相同。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和 dtype 计算不兼容。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

后向接口 C 语言：

```
Status op_cumsum_bwd(const Tensor out_grad,
                     const int axis,
                     Tensor *in_grad);
```

后向接口参数：

out_grad (IN)：表示输出张量的梯度。

axis (IN)：表示输入张量累加的轴，-1 代表按照 1-D 张量求全局的前缀和。

in_grad (OUT)：表示输入张量的梯度

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和 dtype 计算不兼容。

STATUS_OUT_OF_RANGE：表示累加的维度超出输入张量维度。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: shape = [3, 4] */
/* input: data = [[ 0, 1, 2, 3],
                  [ 4, 5, 6, 7],
                  [ 8, 9, 10, 11]] */
Tensor output;
op_cumsum(input, -1, &output);
/* output: shape = [12] */
/* output: data = [0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66] */
op_cumsum(input, 0, &output);
/* output: shape = [3, 4] */
/* output: data = [[0, 1, 2, 3],
                   [4, 6, 8, 10],
                   [12, 15, 18, 21]] */
```

A.2.14 索引操作

A.2.14.1 最大索引

C 语言：

```
Status op_argmax    (const Tensor input,
                     const int axis,
                     const bool keepdim,
                     const string dtype,
```

```
                Tensor *indices);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示求最大元素索引的轴，从 0 开始计数。

keepdim (IN)：是否在输出 Tensor 中保留减小的维度。如果 keepdim 为 true，则输出 Tensor 和 x 具有相同的维度(减少的维度除外，减少的维度的大小为 1)，默认值为 false。

dtype (IN)：输出 Tensor 的数据类型，可选值为 int32，int64，默认值为 int64，将返回 int64 类型的结果。

indices (OUT)：表示索引结果的张量，数据类型为 INT64。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量没有初始化。

STATUS_OUT_OF_RANGE：表示索引维度超出输入张量维度。

示例：

```
/* input: shape = [2, 3, 4] */
/* input: data = [[[5, 8, 9, 5], [0, 0, 1, 7], [6, 9, 2, 4]],
[[5, 2, 4, 5], [4, 7, 7, 9], [1, 7, 0, 6]]] */
Tensor indices;
op_argmax(input, 2, keepdim =ture, &indices);
/* indices: shape = [2, 3] */
/* indices: data = [[2, 3, 1], [0, 3, 1]] */
op_argmax(input, 0, keepdim =ture, &indices);
/* indices: shape = [3, 4] */
/* indices: data = [[0, 0, 0, 0], [1, 1, 1, 1], [0, 0, 0, 1]] */
```

A.2.14.2 最小索引

C 语言：

```
Status op_argmin    (const Tensor input,
                const int axis,
                const bool keepdim,
                const string dtype,
                Tensor *indices);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示求最小元素索引的维度。k>=0 表示第 k+1 维度；k<0 表示倒数第 k 维度。默认值：-1。

keepdim (IN)：是否在输出 Tensor 中保留减小的维度。如果 keepdim 为 true，则输出 Tensor 和 x 具有相同的维度(减少的维度除外，减少的维度的大小为 1)，默认值为 false。

dtype (IN)：输出 Tensor 的数据类型，可选值为 int32，int64，默认值为 int64，将返回 int64 类型的结果。

indices (OUT)：表示索引结果的张量，数据类型为 INT64。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示索引维度超出输入张量维度。

示例：

```
/* input: shape = [2, 3, 4] */
/* input: data = [[[5, 8, 9, 5], [0, 0, 1, 7], [6, 9, 2, 4]],
[[5, 2, 4, 5], [4, 7, 7, 9], [1, 7, 0, 6]]] */
Tensor indices;
op_argmin(input, 2, &indices);
/* indices: shape = [2, 3] */
/* indices: data = [[2, 3, 1], [0, 3, 1]] */
op_argmin(input, 0, &indices);
/* indices: shape = [3, 4] */
/* indices: data = [[0, 0, 0, 0], [1, 1, 1, 1], [0, 0, 0, 1]] */
```

A.2.14.3 排序索引

C 语言：

```
Status op_argsort(const Tensor input,
                  const int axis,
                  const bool descending,
                  Tensor *output,
                  Tensor *indices);
```

参数：

input (IN)：表示输入张量。

axis (IN)：表示待排序的维度，k>=0 表示第 k+1 维度；k<0 表示倒数第 k 维度。默认值：-1。

descending (IN)： true 表示算法按照降序排序，否则按照升序排序。默认值为 true。

output (OUT)：表示排序后的张量，形状、数据类型与张量 input 相同。

indices (OUT)：表示排序后的位置索引张量，形状与张量 input 相同，数据类型为 INT64。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

后向接口 C 语言：

```
Status op_argsort_bwd(const Tensor out_grad,
                      const Tensor indices,
                      Tensor *in_grad);
```

后向接口参数：

out_grad(IN)：表示输出张量的梯度。

indices (IN)：表示排序后的位置索引张量，形状与张量 input 相同，数据类型为 INT64。

in_grad(OUT)：表示输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象不合法。

STATUS_OUT_OF_RANGE：表示 axis 超出输入张量维度。

示例：

```
/* input: shape = [2, 3, 4] */
/* input: data = [[[5, 8, 9, 5], [0, 0, 1, 7], [6, 9, 2, 4]],
                 [[5, 2, 4, 5], [4, 7, 7, 9], [1, 7, 0, 6]]] */
Tensor output, indices;
op_argsort(input, 2, false, &output, &indices);
/* output: data = [[[5, 5, 8, 9], [0, 0, 1, 7], [2, 4, 6, 9]],
                   [[2, 4, 5, 5], [4, 7, 7, 9], [0, 1, 6, 7]]]*/
/* indices: data = [[[0, 3, 1, 2], [0, 1, 2, 3], [2, 3, 0, 1]],
                    [[1, 2, 0, 3], [0, 1, 2, 3], [2, 0, 3, 1]]] */
```

A.2.14.4 Top K索引

C 语言：

```
Status op_topk(const Tensor input,
               const int k,
               const int axis,
               const bool largest,
               const bool sorted
               Tensor *output
               Tensor *indices);
```

参数：

input (IN)：表示输入张量。

k (IN)：表示寻找的最大前 k 项，值必须大于 0 且小于张量 input 最后一维的大小。

axis (IN)：表示 topk 取数据的维度，默认值为 -1。

largest (IN)：指定升序还是降序排列，如果设置为 true，排序算法按照降序的算法排序，否则按照升序排序。默认值为 true。

sorted (IN)：控制返回的 k 个结果是否严格按照有序返回，默认为 true。比如在 GPU 上总是返回有序的 k 个结果。

output (OUT)：表示前 k 个最大项的结果张量，数据类型与张量 input 相同。

indices (OUT)：表示前 k 个最大项的索引张量，数据类型为 INT64。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

示例：

```
/* input: shape = [2, 3, 4] */
/* input: data = [[[5, 8, 9, 5], [0, 0, 1, 7], [6, 9, 2, 4]],
[[5, 2, 4, 5], [4, 7, 7, 9], [1, 7, 0, 6]]] */
Tensor output, indices;
op_topk(input, 2, &output, &indices);
/* indices: shape = [2, 3] */
/* indices: data = [[2, 3, 1], [0, 3, 1]] */
op_topk(input, 0, &output, &indices);
/* indices: shape = [3, 4] */
/* indices: data = [[0, 0, 0, 0], [1, 1, 1, 1], [0, 0, 0, 1]] */
```

A.2.14.5 非零索引

C 语言：

```
Status op_nonzero(const Tensor input,
                  const bool as_tuple,
                  Tensor *indices);
```

参数：

input (IN)：表示输入张量。

as_tuple (IN)：表示输出张量是否是一维张量构成的元组格式。

indices (OUT)：表示索引结果的张量，数据类型为 Tensor 或者 tuple。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：输入张量对象没有初始化。

STATUS_OUT_OF_RANGE：表示不存在非零元素。

STATUS_ALLOC_FAILED：表示输出向量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：内部调用操作出错。

示例：

```
/* input = [[1.0, 0.0, 0.0],
            [0.0, 2.0, 0.0],
            [0.0, 0.0, 3.0]] */
/*as_tuple = false */
Tensor output;
op_nonzero(input, as_tuple , &output);
/*output= [[0, 0],
        [1, 1],
        [2, 2]]*/
/*as_tuple = true */
op_nonzero(input, as_tuple , &output);
/*output= ([[0], [1], [2]], [[0], [1], [2]])*/
```

A.2.15 复数操作

A.2.15.1 复数构建

C 语言：

```
Status op_complex(const Tensor real,
                  const Tensor imag,
                  Tensor *output)
```

参数：

real (IN)：表示需要转换成复数 output 的实部部分。

imag (IN)：表示需要转换成复数 output 虚部部分。

output (OUT)：表示转换后的复数类型的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和要转换的类型不兼容、两个输入张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input: real = [2.25, 3.25], imag = [4.75, 5.75] */
Tensor output;
op_complex(real, imag, &output) ;
/* output: output = [[2.25 + 4.75j], [3.25 + 5.75j]] */
```

A.2.15.2 复数共轭

C 语言：

```
Status op_conj(const Tensor input,
               Tensor *output)
```

参数：

input (IN)：表示要求取复数共轭的输入张量。

output (OUT)：表示对输入张量求取复数共轭后的输出张量。如果 input 元素是实数，则输出张量等于输入张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不兼容。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input = [-2.25 + 4.75j, 3.25 + 5.75j] */
Tensor output;
op_conj(input, &output);
/* output = [-2.25 - 4.75j, 3.25 - 5.75j] */
```

A.2.15.3 获取虚部

C 语言：

```
Status op_imag (const Tensor input,
                Tensor *output)
```

参数：

input (IN)：表示输入张量。

output (OUT)：表示获取输入张量的虚部。如果 input 的元素是实数，则输出张量为 0。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和要转换的类型不兼容。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input = [-2.25 + 4.75j, 3.25 + 5.75j] */
```

```
Tensor output;
op_imag(input, &output);
/* output = [4.75, 5.75] */
```

A.2.15.4 获取实部

C 语言：

```
Status op_real(const Tensor input,
                    Tensor *output)
```

参数：

input (IN)：表示输入张量。

output (OUT)：表示获取输入张量的实部。如果 input 元素是实数，则输出张量等于输入张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和要转换的类型不兼容。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input = [-2.25 + 4.75j, 3.25 + 5.75j] */
Tensor output;
op_real(input, &output);
/* output = [-2.25, 3.25] */
```

A.2.16 信号处理类

A.2.16.1 复数到复数的快速傅里叶变换

C 前向接口：

```
Status op_fft_c2c(const Tensor input,
                  const int* axes,
                  const int axes_len,
                  const FFTNormMode normalization,
                  const bool forward,
                  Tensor *output)
```

参数：

input (IN)：表示需要进行傅里叶变换的输入张量，数据类型为浮点复数。

axes (IN)：表示进行离散傅里叶变换所沿着的维度。

axes_len (IN):表示 axes 数组的长度。

normalization (IN)：表示变换类型，其是枚举类型变量，具体定义为

```
typedef enum _ FFTNormMode {
none,
by_sqrt_n,
```

by_n } FFTNormMode;

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

output (OUT)：表示离散傅里叶变换后的输出张量，数据类型为浮点复数。

返回值：

STATUS_SUCCESS：表示操作成功。170

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

C 后向接口：

```
Status op_fft_c2c_backward(const Tensor grad_out,
                           const int* axes,
                            const int axes_len,
                            const FFTNormMode normalization,
                            const bool forward,
                            Tensor *grad_in)
```

参数：

grad_out (IN)：表示输出张量的梯度，数据类型为浮点复数。

axes (IN)：表示进行离散傅里叶变换所沿着的维度。

axes_len (IN):表示 axes 数组的长度。

normalization (IN)：表示变换类型，其是枚举类型变量，定义和前向 normalization 一样

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

grad_in (OUT)：表示输入张量的梯度，数据类型为浮点复数。

返回值：

STATUS_SUCCESS：表示操作成功。170

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

A.2.16.2 实数到复数的快速逆傅里叶变换

C 语言：

```
Status op_ifft(const Tensor input,
               const int *istride,
               const int istride_len,
```

```
                const TransType transtype,
                const int *ostride,
                const int ostride_len,
                Tensor  *output)
```

参数：

input (IN)：表示需要进行逆傅里叶变换的输入张量。

istride (IN)：表示输入跨度，数组大小为 rank(input)，元素类型为 int。当 istride={1,1,…,rank(input)}时，表示不带输入跨度的快速逆傅里叶变换。

istride_len (IN)：表示 istride 数组的长度。

transtype (IN)：表示变换类型，其是枚举类型变量，具体定义为

```
typedef enum _trans_type {
  C2C,
  R2C,
  C2R,
  R2R
} TransType;
```

ostride (IN)：表示输出跨度，其数组大小为 rank(input)，元素类型为 int。当 ostride = {1,1,…,rank(input)}时，表示不带输出跨度的快速逆傅里叶变换。

ostride_len (IN)：表示 ostride 数组的长度。

output (OUT)：表示逆傅里叶变换后的输出张量，其具有与输入张量相同的形状，但元素类型不一定相同。当 output 等于 input 时，表示逆傅里叶变换的类型为 in-place，否则为 out-of-place。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

C 前向接口：

```
 Status op_fft_r2c(const Tensor input,
                   const int* axes,
                   const int axes_len,
                   const FFTNormMode normalization,
                   const bool forward,
                   const bool onesided,
                   Tensor *output)
```

参数：

input（IN）：表示需要进行傅里叶变换的输入张量，数据类型为浮点实数。

axes（IN）：表示进行快速傅里叶变换所沿着的维度。

axes_len（IN）:表示 axes 数组的长度。

normalization（IN）：表示变换类型，其是枚举类型变量，具体定义为

```
typedef enum _ FFTNormMode {
 none,
 by_sqrt_n,
 by_n } FFTNormMode;
```

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

onesided（IN）：表示复数结果的保留方式。为 true 表示不保留共轭结果，正向和后向都可以节省 1 半的存储空间。为 false 表示保留全部结果。默认为不保留

output（OUT）：表示离散傅里叶变换后的输出张量，数据类型为浮点复数。

返回值：

STATUS_SUCCESS：表示操作成功。172

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

C 后向接口：

```
Status op_fft_r2c_backward(const Tensor grad_out,
                           const Tensor input,
                           const int* axes,
                           const int axes_len,
                           const FFTNormMode normalization,
                           const bool forward,
                           Tensor *grad_in)
```

参数：

grad_out（IN）：表示输出张量的梯度，数据类型为浮点复数。

input（IN）：表示需要进行傅里叶变换的输入张量，数据类型为浮点实数。因为如果正向计算的复数结果不保留共轭，则反向计算逻辑中需要知道输入张量

axes（IN）：表示进行快速傅里叶变换所沿着的维度。

axes_len（IN）:表示 axes 数组的长度。

normalization（IN）：表示变换类型，其是枚举类型变量，定义和前向 normalization 一样

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

grad_in（OUT）：表示输入张量的梯度，数据类型为浮点实数。

返回值：

STATUS_SUCCESS：表示操作成功。172

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

A.2.16.3 复数到实数的快速傅里叶变换

C 前向接口：

```
Status op_fft_c2r(const Tensor input,
                  const int* axes,
                  const int axes_len,
                  const FFTNormMode normalization,
                  const bool forward,
                  const int last_dim_size,
                  Tensor *output)
```

参数：

input (IN)：表示需要进行快速傅里叶变换的输入张量，数据类型为浮点复数。

axes (IN)：表示进行快速傅里叶变换所沿着的维度。

axes_len (IN):表示 axes 数组的长度。

normalization (IN)：表示变换类型，其是枚举类型变量，具体定义为

```
typedef enum _ FFTNormMode {
 none,
 by_sqrt_n,
 by_n } FFTNormMode;
```

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

last_dim_size(IN)：1 个整形值。因为当复数最后 1 维为 N 时，可以变换得到最后 1 维为(N-1)*2 或 (N-1)*2+1 的实数，所以需要指定是得到哪种维度的实数结果

output (OUT)：表示快速傅里叶变换后的输出张量，数据类型为浮点实数。

返回值：

STATUS_SUCCESS：表示操作成功。173

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

C 后向接口：

```
Status op_fft_c2r_backward(const Tensor grad_out,
                           const int* axes,
                           const int axes_len,
```

```
                                const FFTNormMode normalization,
                                const bool forward,
                                Tensor *grad_in)
```

参数：

grad_out (IN)：表示输出张量的梯度，数据类型为浮点实数。

的复数结果不保留共轭，则反向计算逻辑中需要知道输入张量

axes (IN)：表示进行离散傅里叶变换所沿着的维度。

axes_len (IN):表示 axes 数组的长度。

normalization (IN)：表示变换类型，其是枚举类型变量，定义和前向 normalization 一样

forward(IN)：表示傅里叶变换方式。为 true 表示正变换，为 false 表示逆变换。默认正变换

grad_in (OUT)：表示输入张量的梯度，数据类型为浮点复数。

返回值：

STATUS_SUCCESS：表示操作成功。174

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH: 表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

A.2.17 线性代数操作

A.2.17.1 矩阵乘法

C 语言：

```
Status op_matmul(const Tensor mat1,
                 const Tensor mat2,
                 Tensor *out);
```

参数：

mat1 (IN)：第一个输入张量。

mat2 (IN)：第二个输入张量。

out (OUT)：输出两个张量的乘积。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

STATUS_DIMENSIONS_MISMATCH: 表示张量维度不匹配等。

A.2.17.2 向量内积

C 语言：

```
Status op_dot(const Tensor vec1,
              const Tensor vec2,
              Tensor *out);
```

参数：

vec1 (IN)：第一个输入张量。

vec2 (IN)：第二个输入张量。

out (OUT)：输出张量 vec1 和张量 vec2 的内积。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

后向接口 C 语言：

```
Status op_dot_bwd1(const Tensor out_grad,
              const Tensor vec1,
              const Tensor vec2,
              Tensor *in_grad1);
```

后向接口参数：

out_grad(IN):输出张量的梯度。

Vec1 (IN)：第一个输入张量。

vec2 (IN)：第二个输入张量。

in_grad1 (OUT)：第一个输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

后向接口 C 语言：

```
Status op_dot_bwd2(const Tensor out_grad,
              const Tensor vec1,
              Tensor *in_grad2);
```

后向接口参数：

out_grad(IN):输出张量的梯度。

vec1(IN)：第一个输入张量。

in_grad2(OUT)：第二个输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

A.2.17.3 LU矩阵分解

C 语言：

```
Status op_lu(const Tensor mat,
             Tensor *lu,
             Tensor *ipiv);
```

参数：

mat (IN)：描述待分解矩阵。

lu (OUT)：描述分解结果。

ipiv (OUT)：描述主元交换过程的张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

A.2.17.4 cholesky矩阵分解

C 语言：

```
Status op_cholesky(const char upper,
                   const Tensor mat,
                   Tensor *out);
```

参数：

upper (IN)：表示输入张量 mat 的值是上三角有效还是下三角有效。

mat (IN)：描述对称正定矩阵。

out (OUT)：输出时基于 upper 的值 inner-most 的值对应上三角或者下三角矩阵，且其维度与输入矩阵维度保持一致。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

A.2.17.5 QR矩阵分解

C 语言：

```
Status op_qr(const Tensor mat,
             const string mode,
             Tensor *Q,
             Tensor *R);
```

参数：

mat (IN)：描述待分解矩阵。

mode(IN)：控制正交三角分解的行为，默认是 reduced ，假设 x 形状应为 [*, M, N] 和 K = min(M, N)：如果 mode = "reduced" ，则 Q 形状为 [*, M, K] 和 R 形状为 [*, K, N] ；如果 mode = "complete" ，则 Q 形状为 [*, M, M] 和 R 形状为 [*, M, N] ；如果 mode = "r" ，则不返回 Q，只返回 R 且形状为 [*, K, N] 。

Q (OUT)：描述输出正交矩阵。

R (OUT)：描述输出上三角矩阵，其维度特征描述同输出张量 Q。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

A.2.17.6 SVD奇异值分解

C 语言：

```
Status op_svd(const Tensor mat,
              const bool format,
              Tensor *U,
              Tensor *S,
              Tensor *V);
```

参数：

mat (IN)：描述输入待分解矩阵。

format(IN)：当 format 为 true 时，返回张量 U 和 V 仅包含 min(n,m)个正交列一维张量，否则返回张量 U 和 V 分别包含 n 和 m 个正交列一维张量。

U (OUT)：分解张量 U。

S(OUT)：奇异值组成的对象张量。

V (OUT)：分解张量 V。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

示例：

A.2.17.7 线性方程组求解

C 语言：

```
Status op_solve(const Tensor mat,
                const Tensor rhs,
                const bool adjoint,
```

```
                    Tensor *x);
```

参数：

mat (IN)：描述系数矩阵。

rhs (IN)：描述线性方程组的右端项。

adjoint (IN)：描述使用系数矩阵 mat 求解还是使用其伴随矩阵求解(block-wise adjoint matrix)。

x (OUT)：描述输出解张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

示例：

```
/* mat:[[1.0,2.0,3.0],[4.0,5.0,6.0],[7.0,8.0,9.0]]*/
/* rhs: [1.0, 2.0, 3.0]*/
/* adjoint : false*/
Tensor x;
op_solve(mat, rhs, adjoint , &x);
/* x:[1/3, -2/3, 0]*/
```

A.2.17.8 最小二乘

C 语言：

```
Status op_lstsq(const Tensor mat,
                const Tensor rhs,
                const float prec,
                Tensor *x);
```

参数：

mat (IN)：描述系数矩阵的输入张量。

rhs (IN)：右端项。

Prec(IN)：描述输入张量 mat 的奇异值截止比。

x (OUT)：描述输出最小二乘解。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

示例：

```
/* mat:[[1.0,2.0,3.0],[4.0,5.0,6.0],[7.0,8.0,9.0]]*/
/* rhs: [1.0, 2.0, 3.0]*/
/* Prec: 1e-10*/
 Tensor x;
 op_lstsq(mat, rhs, Prec, &x);
/* x:[[-0.05555556,  0.11111111,  0.27777778],[],2,[1.68481034e+01, 1.06836951e+00,
3.33475287e-16]]*/
```

A.2.17.9 矩阵求逆

C 语言：

```
Status op_inverse(const Tensor mat,
                  Tensor *inv);
```

参数：

mat (IN):描述待求逆的矩阵。

inv (out)：输入张量的逆张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

示例：

```
/* mat:[[1,2],[2,1]]*/
Tensor inv;
op_inverse(mat, &inv);
 /* inv:[[-0.3333, -0.6667],
       [-0.6667, -0.3333]]*/
```

A.2.17.10 求特征值以及特征向量

C 语言：

```
Status op_eig(const Tensor mat,
              Tensor *w,
              Tensor *v);
```

参数：

mat (IN): 待计算特征值和右端特征向量的方阵。

w (out)：描述返回特征值的张量。

v (out): 描述返回特征向量的张量，且其第 i 列 v[:,i]与第 i 个特征值相对应。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_INVALID_ARGUMENT：表示其他参数出错情况。

示例：

```
/* mat:[[1,2],[2,1]]*/
Tensor w,v;
op_eig(mat, &w, &v);
/* w: [-1,3]        */
/* v:[[-1, 1],
      [1, 1]]     */
```

A.2.17.11 矩阵范数

C 语言：

```
Status op_norm( const Tensor input,
                const char *normtype,
                const int *axis,
                const int axis_len,
                const bool keep_dims,
                Tensor *output)
```

参数：

input (IN)：表示输入张量。

normtype (IN)：表示范数计算的类别。

axis (IN)：轴数组。

axis_len (IN)：表示 axis 数组的长度。

keep_dims (IN)：如果为 true，则保留输入张量的维度。否则，输入的维度将小于输入的维度。

output (OUT)：表示与输入张量具有相同类型的输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_UNINITIALIZED_OBJECT：表示输入张量对象未初始化。

STATUS_TYPE_MISMATCH：表示输入张量对象的类型和输出张量的类型不匹配。

STATUS_ALLOC_FAILED：表示输出张量分配空间不足。

STATUS_INVALID_ARGUMENT：表示其他参数不合法。

STATUS_INTERNAL_ERROR：表示内部的调用操作出错。

示例：

```
/* input = [[1, 2, 3], [4, 5, 6]] */
/* axis = [] */
Tensor output;
op_norm(input, '2', axis, 0, false, &output);
/* output = 9.53939 */
/* axis = [] */
```

```
op_norm(input, '1', axis, 0, false, &output);
/* output = 21.0 */
/* axis = [0] */
op_norm(input, '1', axis, 1, false, &output);
/* output = [5., 7., 9.] */
/* axis = [1] */
op_norm(input, '1', axis, 1, false, &output);
/* output = [6., 15] */
/* axis = [1] */
op_norm(input, '1', axis, 1, true, &output);
/* output = [[6.], [15]] */
```

A.2.17.12 线性操作

C 语言：

```
Status op_linear( const Tensor x,
                  const Tensor weight,
                  const Tensor bias,
                  Tensor *y);
```

参数：

x (IN)：表示输入张量。

weight (IN)：表示线性操作的权重。

bias(IN)：表示偏置。

y (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

后向接口 C 语言：

```
Status op_linear_bwd_in( const Tensor out_grad,
                  const Tensor weight,
                  Tensor *in_grad);
```

后向接口参数：

out_grad (IN)：输出张量的梯度。

weight (IN)：表示线性操作的权重。

in_grad (OUT)：表示输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

后向接口 C 语言：

```
Status op_linear_bwd_weight( const Tensor out_grad,
                             const Tensor input,
                             Tensor *weight_grad);
```

后向接口参数：

out_grad (IN)：输出张量的梯度。

input (IN)：表示输入张量。

weight_grad (OUT)：表示权重张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

示例：

```
/* x: [[1.0, 2.0, 3.0],
      [4.0, 5.0, 6.0]]   */
/* weight: [1.0, 2.0, 3.0]  */
/* bias: [0.1]  */
Tensor y;
op_linear(x, weight, bias, &y);
/* y:  [[14.1],
       [32.1]] */
```

A.2.17.13 双线性操作

C 语言：

```
Status op_bilinear( const Tensor x,
                    const Tensor y,
                    const Tensor weight,
                    const Tensor bias,
                    Tensor *z);
```

参数：

x (IN)：表示第一个输入张量。

y (IN)：表示第二个输入张量。

weight (IN)：表示线性操作的权重。

bias(IN)：表示偏置。

z (OUT)：表示输出张量。

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

后向接口 C 语言：

```
Status op_bilinear_bwd_in1( const Tensor out_grad,
                    const Tensor y,
                    const Tensor weight,
                    Tensor *in_grad1);
```

后向接口参数：

out_grad（IN）：表示第一个输入张量。

y（IN）：表示第二个输入张量。

weight（IN）：表示线性操作的权重。

in_grad1（OUT）：表示第一个输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

后向接口 C 语言：

```
Status op_bilinear_bwd_in2( const Tensor out_grad,
                    const Tensor x,
                    const Tensor weight,
                    Tensor *in_grad2);
```

后向接口参数：

out_grad（IN）：表示第一个输入张量。

x（IN）：表示第一个输入张量。

weight（IN）：表示线性操作的权重。

in_grad2(OUT)：表示第二个输入张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

后向接口 C 语言：

```
Status op_bilinear_bwd_weight( const Tensor out_grad,
                    const Tensor x,
                    const Tensor y,
                    Tensor *weight_grad);
```

后向接口参数：

out_grad（IN）：表示第一个输入张量。

x (IN)：表示第一个输入张量。

y (IN)：表示第二个输入张量。

weight_grad(OUT)：表示权重张量的梯度。

后向接口返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

示例：

```
/* x: [[0.8277, 0.7097, 0.3228],
       [0.5277, 0.1352, 0.5379]]   */
/* [[0.7230, 0.1641, 0.0972, 0.8643],
       [0.1888, 0.2875, 0.2289, 0.9244]] */

/* weight: [[[0.2222, 0.6071, 0.0856, 0.0199],
         [0.1799, 0.3981, 0.1620, 0.8210],
         [0.5154, 0.8984, 0.9911, 0.6962]],
        [[0.2050, 0.4599, 0.3099, 0.0036],
         [0.9957, 0.2217, 0.5122, 0.8350],
         [0.1328, 0.0032, 0.2324, 0.3096]],
        [[0.1380, 0.1614, 0.3526, 0.8682],
         [0.1394, 0.3380, 0.3199, 0.3564],
         [0.2648, 0.9249, 0.8641, 0.0150]]]  */
/* bias: [0.7517, 0.0115, 0.8519]  */
Tensor z;
op_bilinear(x, y, weight, bias, &z);
/* z:  [[2.0348, 1.4333, 2.0994],
       [1.6731, 0.4918, 1.7111]]*/
```

A.2.18 插值操作

C 语言：

```
Status op_interpolate(const Tensor input,
                      const int* size,
                      const float* scale_factor,
                      const char *mode,
                      Tensor *output);
```

参数：

input (IN)：表示待进行插值的输入张量。

size (IN)：表示输出张量的尺寸。

scale_factor(IN)：表示输出张量尺寸的缩放因子。

mode (IN)：表示采用的采样算法。

output (OUT)：表示输出张量

返回值：

STATUS_SUCCESS：表示操作成功。

STATUS_TYPE_MISMATCH：表示参数的数据类型不一致。

STATUS_DIMENSIONS_MISMATCH：表示维度不匹配。

示例：

```
/* input:  [[[[1.0, 2.0],
          [3.0, 4.0]]]]
 */
/* size: [4, 4]  */
/* mode = “bilinear” */
Tensor output;
op_interpolate (input, size, NULL, mode, &output);
/* output:  [[[[1.0000, 1.2500, 1.7500, 2.0000],
          [1.5000, 1.7500, 2.2500, 2.5000],
          [2.5000, 2.7500, 3.2500, 3.5000],
          [3.0000, 3.2500, 3.7500, 4.0000]]]] */
```

参 考 文 献

[1] IEEE 2941.1-2022 IEEE Standard for Operator Interfaces of Artificial Intelligence

[2] 杨超.“人工智能算子接口标准化研究.”知网，2020，https://mall.cnki.net/magazine/Article/DKJS202003002.htm