

IEEE Standard for Operator Interfaces of Artificial Intelligence

IEEE Computer Society

Developed by the
Data Compression Standards Committee

IEEE Std 2941.1™-2022

STANDARDS

IEEE Standard for Operator Interfaces of Artificial Intelligence

Developed by the

Data Compression Standards Committee
of the
IEEE Computer Society

Approved 8 November 2022

IEEE SA Standards Board

Abstract: A set of operator interfaces frequently used in artificial intelligence (AI) applications is defined in this standard, where the AI operators refer to the standard building blocks and primitives for performing basic AI operations. The functionality and the specific input and output operands of an AI operator are discussed, as well as both generality and efficiency. Various types of operators, such as those related to basic mathematics, neural network, and machine learning, are highlighted.

Keywords: artificial intelligence, AI, basic mathematics, IEEE 2941.1™, neural network, machine learning, operators

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2023 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 24 March 2023. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-9533-2 STD26028
Print: ISBN 978-1-5044-9534-9 STDPD26028

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE is the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s).

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents.**

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and subcommittees of the IEEE SA Board of Governors are not able to provide an instant response to comments, or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the [IEEE SA myProject system](#).¹ An IEEE Account is needed to access the application.

Comments on standards should be submitted using the [Contact Us](#) form.²

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and

¹ Available at: <https://development.standards.ieee.org/myproject-web/public/view.html#landing>.

² Available at: <https://standards.ieee.org/content/ieee-standards/en/about/contact/index.html>.

adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#).³ For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#).⁴ Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

Patents

IEEE Standards are developed in compliance with the [IEEE SA Patent Policy](#).⁵

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

³ Available at: <https://ieeexplore.ieee.org/browse/standards/collection/ieee>.

⁴ Available at: <https://standards.ieee.org/standard/index.html>.

⁵ Available at: <https://standards.ieee.org/about/sasb/patcom/materials.html>.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

IMPORTANT NOTICE

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. IEEE Standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

Participants

At the time this IEEE standard was completed, the AI Model Representation, Compression, Distribution, and Management Working Group had the following membership:

Yonghong Tian, *Chair*
Chao Yang, *Vice Chair*

At the time this IEEE standard was completed, the 2941.1 Sub-Working Group had the following membership:

Chao Yang, *Chair*
Yonghong Tian, Xiaoguang Hu, Chun Fan, *Vice Chairs*
Xiaoxu Luan, *1st Secretary*
Haijun Feng, *2nd Secretary*
Min Li, *Technical Editor*
Lian Duan, *Technical Sub-editor*

Yulong Ao
Xiaoqi Cao
Deliang Chen
Kai Chen
Ruibo Fan
Tiezhu Gao
He Guan
Zhenhua Guo
Shuai Hu
Jiajun Jiang

Xiaolin Jiang
Hui Li
Kesen Li
Xiaoru Li
Yurui Li
Ziyi Li
Yiqun Liu
Rengang Li
Yanjuan Ma
Chen Qian
Li Wang

Xinmin Wang
Feng Wu
Geng Wu
Bo Zhang
Xiaofei Zhang
Xingcheng Zhang
Haiying Zhao
Qiao Zhao
Shengyuan Zhou
Yaqian Zhao

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Diego Chiozzi
Piotr Karocki
Alexis Kokkonos
Jun Li
Ting Li
Xiaoru Li
Guojie Luo

Siwei Ma
Cesar Negri
Joanna Olszewska
Sivaraman P
R. K. Rannow
Pablo Rivas Perea
Barnaby Simkin
Walter Struppler

Yonghong Tian
Yaowei Wang
Yu Wang
Feng Wu
Lu Yu
Yu Yuan
Yuyi Zhang

When the IEEE SA Standards Board approved this standard on 8 November 2022, it had the following membership:

David J. Law, *Chair*
Ted Burse, *Vice Chair*
Gary Hoffman, *Past Chair*
Konstantinos Karachalios, *Secretary*

Edward A. Addy
Ramy Ahmed Fathy
J. Travis Griffith
Guido R. Hiertz
Yousef Kimiagar
Joseph L. Koepfinger*
Thomas Koshy
John D. Kulick

Johnny Daozhuang Lin
Kevin Lu
Daleep C. Mohla
Andrew Myles
Damir Novosel
Annette D. Reilly
Robby Robson
Jon Walter Rosdahl

Mark Siira
Dorothy V. Stanley
Lei Wang
F. Keith Waters
Karl Weber
Sha Wei
Philip B. Winston
Daidi Zhong

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 2941.1-2022, IEEE Standard for Operator Interfaces of Artificial Intelligence.

This standard defines a series of interfaces for AI operators, which are the cornerstone of mainstream AI applications and frameworks. The interfaces of various types of operators are specified, including basic mathematical operators, neural network operators, and machine learning operators. The design of the interfaces considers both generality and efficiency. The operator interface definition does not bind to any specific programming language, but the C programming language reference definition is proposed to provide a clearer interface form and avoid ambiguity.

Contents

1. Overview	12
1.1 Scope	12
1.2 Purpose	12
1.3 Word usage	13
2. Normative references	13
3. Definitions, acronyms, and abbreviations	13
3.1 Definitions	13
3.2 Acronyms and abbreviations	14
4. Symbols and operators	15
4.1 Arithmetic operators	15
4.2 Logical operator	16
4.3 Relational operators	16
4.4 Bitwise operators	16
5. General principles	17
5.1 Starting subscript	17
5.2 Order of parameters	17
5.3 Programming language	17
5.4 Broadcasting	17
5.5 Error handling	17
5.6 Interface consistency between dense and sparse tensors	17
5.7 Functional consistency and hierarchical difference	18
6. Data structure	18
6.1 Element data type	18
6.2 Shape information	18
6.3 Layout information	18
6.4 Device information	19
6.5 Other extensions	19
7. Interfaces for basic mathematical operators	19
7.1 Tensor creation and destruction	19
7.2 Query and inspection	29
7.3 Tensor conversion	32
7.4 Arithmetic operations	43
7.5 Comparison operation	49
7.6 Logical operation	53
7.7 Bitwise operation	55
7.8 Power function	58
7.9 Rounding operation	60
7.10 Trigonometric function	61
7.11 Hyperbolic function	64
7.12 Exponential and logarithmic function	67
7.13 Reduction	70
7.14 Indexing	71
7.15 Complex	74
7.16 Signal processing	76
7.17 Linear algebra	77

8. Interfaces for neural network operators	83
8.1 Activation function	83
8.2 Loss function	100
8.3 Regularization function	110
8.4 Normalization function	112
8.5 Pooling function	120
8.6 Convolution function	126
8.7 Evaluation function	137
8.8 Recurrent network function	138
8.9 Encoding function	154
8.10 Distance function	156
8.11 Visual function	156
8.12 Optimizer function	158
9. Interfaces for machine learning operators	166
9.1 Linear regression algorithm	166
9.2 Logistic regression algorithm	168
9.3 Decision tree classifier (DTC)	170
9.4 Decision tree regressor (DTR)	172
9.5 Random forest classifier (RFC)	174
9.6 Random forest regressor (RFR)	176
9.7 Gaussian naïve Bayesian algorithm (GNB)	178
9.8 Linear discriminant analysis (LDA)	180
9.9 Principal component analysis (PCA)	182
9.10 K-nearest neighbor algorithm (KNN)	184
9.11 Support vector machine (SVM) algorithm	187
9.12 K-means clustering (Kmeans)	191
Annex A (informative) C reference of operator interfaces	194
A.1 Data structure	194
A.2 C interfaces for basic mathematical operators	196
A.3 C interfaces for neural network operators	245
A.4 C interfaces for machine learning operators	303
Annex B (informative) Bibliography	326

IEEE Standard for Operator Interfaces of Artificial Intelligence

1. Overview

1.1 Scope

This standard defines a set of operator interfaces frequently found in artificial intelligence (AI) applications, where the AI operators refer to the standard building blocks and primitives for performing basic AI operations. It defines the functionality and the specific input and output operands of an AI operator. It considers both generality and efficiency and covers various types of operators, such as those related to basic mathematics, neural network, and machine learning. The defined operator interfaces serve as a reference for vendors to implement and use AI operators.

1.2 Purpose

This standard standardizes the development interfaces for commonly used AI operators, which helps to reduce the complexity of AI software and hardware adaptation, improve the efficiency and portability of AI software development, promote the evaluation and promotion of AI hardware, and promote the benign development of AI software and hardware ecology.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).^{6, 7}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 2941TM-2021, IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management.^{8, 9}

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.¹⁰

Definitions in IEEE Std 2941-2021,¹¹ as well as the following terms and definitions, are applicable to this document.

accelerating processor for artificial intelligence: An integrated circuit component with a computing micro-architecture adapted to artificial intelligence algorithms that can complete the computing processing of artificial intelligence applications.

batch: A part of the training sample.

broadcasting: A method that can be used when performing element-wise operations on two tensors of different shapes.

⁶ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements, *must* is used only to describe unavoidable situations.

⁷ The use of *will* is deprecated and cannot be used when stating mandatory requirements, *will* is only used in statements of fact.

⁸ The IEEE standards or products referred to in Clause 2 are trademarks owned by The Institute of Electrical and Electronics Engineers, Incorporated.

⁹ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<https://standards.ieee.org/>).

¹⁰ *IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>. An IEEE Account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

¹¹ Information on references can be found in Clause 2.

dense tensor: A tensor whose elements are all or most nonzero values.

NOTE—Uncompressed storage are used to store dense tensors, that is, dense storage, which stores all elements of the tensor in a certain order.¹²

high-dimensional tensor: A tensor with dimensions higher than 2.

model inference: The process of using an AI model to process data and obtain prediction results.

model training: The process of establishing or improving the parameters of an AI model based on learning algorithm and training data.

sparse tensor: A tensor containing a proportion of zero-valued elements.

NOTE—Compressed storage methods are used to store sparse tensors, that is, sparse storage, which only stores nonzero elements of the tensor.

tensor: A multidimensional array consisting of elements of a single type.

3.2 Acronyms and abbreviations

1-D	one-dimensional
2-D	two-dimensional
3-D	three-dimensional
4-D	four-dimensional
5-D	five-dimensional
AI	artificial intelligence
ASIC	application-specific integrated circuit
AUC	area under the curve
BN	batch normalization
CPU	central processing unit
DTC	decision tree classifier
DTR	decision tree regressor
EIG	eigenvalue
FPGA	field programmable gate array
GN	group normalization
GNB	Gaussian naïve Bayesian
GPU	graphic processing unit
GRU	gated recurrent unit
IN	instance normalization
KMeans	K-means clustering

¹² Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

KNN	K-nearest neighbor
LDA	linear discriminant analysis
LN	layer normalization
LRN	local response normalization
LSR	label-smoothing regularization
LSTM	long short-term memory
NPU	neural-network processing unit
PCA	principal component analysis
ReLU	rectified linear unit
RFC	random forest classifier
RFR	random forest regressor
RNN	recursive neural network
SGD	stochastic gradient descent
SVM	support vector machine

4. Symbols and operators

4.1 Arithmetic operators

The following arithmetic operators are adapted to this document:

+	plus
−	minus (binary operator) or neg (unary prefix operator)
×	multiply
a^b	power, takes the power of a with b
/	division, without truncation or rounding
$\frac{a}{b}$	division, without truncation or rounding
	absolute value, when $x \geq 0$, $ x = x$, when $x < 0$, $ x = -x$
abs()	absolute value
sign()	take sign $\text{sign}(x) = \begin{cases} 1; x \geq 0 \\ -1; x < 0 \end{cases}$
$\sqrt{\quad}$	square root
$\sum_{i=a}^b f(i)$	cumulative sum of the function $f(i)$ when the independent variable i takes all the integer values from a to b (including b)

$\log_{10}$	denary logarithm
$\log_2$	binary logarithm
$\lfloor \rfloor$	round down
exp	exponential function based on natural constant e
*	conjugate operation

4.2 Logical operators

The following logical operators are adapted to this document:

$\parallel$	logical OR
$\&\&$	logical AND
$!$	logical NOT

4.3 Relational operators

The following relational operators are adapted to this document:

$>$	greater than
$>=$	greater than or equal to
$<$	less than
$<=$	less than or equal to
$=$	equal to
$!=$	not equal to
$\max(...)$	maximum value of all the value
$\min(...)$	minimum value of all the value

4.4 Bitwise operators

The following bitwise operators are adapted to this document:

$\&$	AND
$ $	OR
$\sim$	neg
$a \gg b$	move a expressed in the form of two's complement integer rightward for b places. This operation is defined only when b takes a positive number.
$a \ll b$	move a expressed in the form of two's complement integer leftward for b places. This operation is defined only when b takes a positive number.

5. General principles

5.1 Starting subscript

Numbering and counting should start at 0.

5.2 Order of parameters

The input parameters should precede the output parameters in the operator interfaces.

5.3 Programming language

The operator interfaces can be implemented with a variety of programming languages. Please refer to Annex A for the C language reference definition.

5.4 Broadcasting

The element-wise operation does not require that the two input tensors have the same shape, and the tensor with a smaller dimension size is expanded to match the tensor with a larger dimension size to complete the calculation.

5.5 Error handling

The operator should exit directly for unrepairable errors during execution and return error codes for repairable errors. The caller can decide how to handle the errors. The error code should be one of the following values:

- No errors (STATUS_SUCCESS)
- Type mismatch (STATUS_TYPE_MISMATCH)
- Dimension mismatch (STATUS_DIMENSIONS_MISMATCH)
- Object not initialized (STATUS_UNINITIALIZED_OBJECT)
- Illegal parameters (STATUS_INVALID_ARGUMENT)
- Out of memory (STATUS_ALLOC_FAILED)
- Out of range (STATUS_OUT_OF_RANGE)
- Other internal errors (STATUS_INTERNAL_ERROR)

5.6 Interface consistency between dense and sparse tensors

The construction of sparse and dense tensors should use different interfaces, and various operations based on the two types of tensors should use a unified interface.

5.7 Functional consistency and hierarchical difference

The functionality of operators defined in this standard is consistent with the same operators in IEEE Std 2941-2021. Compared with IEEE Std 2941-2021, this standard targets at the functional and parameter requirements of the programming and developing interfaces of AI operators, which is at a lower level and is a supplement to the definition of operators in IEEE Std 2941-2021. More specific parameter lists are provided for operators with the consideration of forward, backward, and inference stage; interfaces based on C language are given in Annex A for clearer specification.

6. Data structure

The tensor is defined as a multidimensional array composed of elements with the same data type, including the aspects outlined in 6.1.

6.1 Element data type

The data type is used to describe the type of elements in the tensor, and the following types should be supported:

- Unsigned integer: 8-bit, 16-bit, 32-bit, 64-bit
- Signed integer: 8-bit, 16-bit, 32-bit, 64-bit
- Floating-point real number: 16-bit, 32-bit, 64-bit
- Floating-point complex number: (32+32)-bit, (64+64)-bit
- Boolean
- String

6.2 Shape information

The shape information is used to describe the number of a tensor's dimensions and the size of each dimension. If the number of the tensor's dimensions is n and the corresponding size of each dimension is $d_0, d_1, d_2, \dots, d_{(n-1)}$, the shape of the tensor can be expressed as $[d_0, d_1, d_2, \dots, d_{(n-1)}]$, the leftmost is the first dimension, and the rightmost is the n th dimension. For example, the number of a scalar's dimensions is 0, and its shape is $[]$; the number of a vector's dimensions is 1, and its shape is $[d_0]$; the number of a matrix's dimension is 2, and its shape is $[d_0, d_1]$.

6.3 Layout information

The layout information is used to describe the storage format of the tensor and the logical order of each dimension. Storage formats include dense format and sparse format. The logical order refers to the priority order of each dimension when the tensor traverses to read and store specific data in memory.

6.4 Device information

The device information is used to describe the type and the device id used to store tensor data. The device types include CPU, GPU, FPGA, ASIC, NPU, or any other devices that support AI operations. If there are multiple devices of the same type, they can be distinguished by specifying the device ID.

6.5 Other extensions

The other extensions are used to provide developers with custom extended functions, such as memory management and other related information.

7. Interfaces for basic mathematical operators

7.1 Tensor creation and destruction

7.1.1 Create dense tensor by copying existing data

7.1.1.1 Function

Create a tensor. If the initialization array is not empty, copy the values of the initialization array to the tensor's space; otherwise create an empty tensor.

7.1.1.2 Interface parameters

The interface shall conform to Table 1, and the C code example (op_create_tensor) is shown in A.2.1.1.

Table 1—Parameter list for creating dense tensor by copying existing data

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Output tensor	Output	Required	The newly created tensor.
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
Initialization array	Input	Optional	The array to initialize the tensor data; if empty, then a tensor full of zero is create; default can be empty.
Length	Input	Optional	The amount of data to be copied, measured in bytes. The insufficient part shall be supplemented with 0; default can be 0.

7.1.1.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of the tensors were inconsistent.

Illegal parameters: The dimensions of the weight and bias tensors did not match the input and output tensors.

Out of memory: Insufficient space was allocated to create a tensor.

Dimension mismatch: Tensor and initialization tensor dimensions did not match.

Other internal errors: An error occurred in the internal call operation.

7.1.2 Create dense tensor by wrapping existing data

7.1.2.1 Function

Create a tensor by wrapping an existing array of data.

7.1.2.2 Interface parameters

The interface shall conform to Table 2, and the C code example (op_wrap_tensor) is shown in A.2.1.2.

Table 2—Parameter list for creating dense tensor by wrapping existing data

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Initialization array	Input	Required	The array to initialize the tensor.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.

7.1.2.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The dimensions of the weight and bias tensor did not match the input output tensors.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.2.4 Others

The caller needs to ensure that the device type of the initialization array is consistent with the expected tensor and that the size of the array is consistent with the tensor.

7.1.3 Create all-zero dense tensor

7.1.3.1 Function

Create a dense tensor with zero data.

7.1.3.2 Interface parameters

The interface shall conform to Table 3, and the C code example (op_zeros) is shown in A.2.1.3.

Table 3—Parameter list for creating all-zero dense tensor

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Output tensor	Output	Required	Newly created tensor
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.

7.1.3.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.4 Create dense tensor with the specified value

7.1.4.1 Function

Create a dense tensor with elements being a specified value.

7.1.4.2 Interface parameters

The interface shall conform to Table 4, and the C code example (op_full) is shown in A.2.1.4.

Table 4—Parameter list for creating dense tensor with specified value

Parameter	Type	Optional/Required	Description
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
Tensor shape	Input	Optional	An array of dimension sizes.
Initialization value	Input	Optional	The value to initialize the tensor.

7.1.4.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.5 Create uninitialized dense tensor

7.1.5.1 Function

Create a tensor in accordance with the specified type, device, and shape. The data of the created tensor are the original data content of the requested memory, un-zeroed or uninitialized, and may be any legal value.

7.1.5.2 Interface parameters

The interface shall conform to Table 5, and the C code example (op_empty) is shown in A.2.1.5.

Table 5—Parameter list for creating uninitialized tensor

Parameter	Type	Optional/Required	Description
Shape	Input	Required	The dimensions of the new tensor.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type of the new tensor; default can be CPU.

7.1.5.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.6 Create dense tensor with continuous memory

7.1.6.1 Function

If the data of the input tensor are stored continuously in memory, this continuous input tensor is returned without processing; otherwise the data of the current tensor are copied to a contiguous section of memory and returned.

7.1.6.2 Interface parameters

The interface shall conform to Table 6, and the C code example (op_contiguous) is shown in A.2.1.6.

Table 6—Parameter list for creating continuous memory tensor

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor.
Output tensor	Output	Required	The output tensor with continuous memory.

7.1.6.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: An internal call operation error occurred.

7.1.7 Create dense tensor with uniform distribution

7.1.7.1 Function

Each element of the created dense tensor is randomly generated from the corresponding uniform distribution.

7.1.7.2 Interface parameters

The interface shall conform to Table 7, and the C code example (op_rand_uniform) is shown in A.2.1.7.

Table 7—Parameter list for creating dense tensor with uniform distribution

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
Minimum value	Input	Optional	The minimum value of the uniform distribution; default can be 0.
Maximum value	Input	Optional	The maximum value of the uniform distribution; default can be 1.
Random seed	Input	Optional	The seed for generating the random number; default can be 0.

7.1.7.3 Interface returns

No errors: The operation was successful.
Illegal parameters: The parameters were wrong.
Out of memory: Insufficient space was allocated to create a tensor.
Other internal errors: An internal call operation error occurred.

7.1.8 Create dense tensor with normal distribution

7.1.8.1 Function

Each element of the created dense tensor is randomly generated from the corresponding normal distribution.

7.1.8.2 Interface parameters

The interface shall conform to Table 8, and the C code example (op_rand_normal) is shown in A.2.1.8.

Table 8—Parameter list for creating dense tensor with normal distribution

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
Mean value	Input	Optional	The mean value of the normal distribution; default can be 0.
Standard deviation	Input	Optional	The standard deviation of the normal distribution; default can be 1.
Random seed	Input	Optional	The seed for generating the random number; default can be 0.

7.1.8.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.9 Create dense tensor with bernoulli distribution

7.1.9.1 Function

Create a dense tensor with bernoulli distribution.

7.1.9.2 Interface parameters

The interface shall conform to Table 9, and the C code example (op_rand_bernoulli) is shown in A.2.1.9.

Table 9—Parameter list for creating dense tensor with bernoulli distribution

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor with probability values.
Output tensor	Output	Required	A tensor containing 0 or 1.

7.1.9.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.10 Create tensor with multinomial distribution

7.1.10.1 Function

Create a tensor with each element generated from the multinomial distribution.

7.1.10.2 Interface parameters

The interface shall conform to Table 10, and the C code example (op_multinomial) is shown in A.2.1.10.

Table 10—Parameter list for creating tensor with multinomial distribution

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor with probability values.
Output tensor	Output	Required	A random tensor with the same shape and data type as the input.
Sampling numbers	Input	Optional	The numbers of sampling; default can be 0.
Replacement	Input	Optional	Whether it is replaceable sampling; default can be false.

7.1.10.3 Interface returns

No errors: The operation was successful.
Illegal parameters: The parameters were wrong.
Out of memory: Insufficient space was allocated to create a tensor.
Other internal errors: An internal call operation error occurred.

7.1.11 Create one-dimensional dense tensors of randperm

7.1.11.1 Function

Create a 1-D tensor with random permutation values from 0 to Length – 1.

7.1.11.2 Interface parameters

The interface shall conform to Table 11, and the C code example (op_randperm) is shown in A.2.1.11.

Table 11—Parameter list for creating a 1-D dense tensor of randperm

Parameter	Type	Optional/Required	Description
Length	Input	Required	The length of the tensor, and upper bound of the range, shall be greater than 0.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer and float; default can be integer.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.

7.1.11.3 Interface returns

No errors: The operation was successful.
Illegal parameters: The parameters were wrong.
Out of memory: Insufficient space was allocated to create a tensor.
Other internal errors: An internal call operation error occurred.

7.1.12 Create dense tensor with a sequence of numbers

7.1.12.1 Function

Create a 1-D dense tensor with spaced values within a given interval.

7.1.12.2 Interface parameters

The interface shall conform to Table 12, and the C code example (op_range) is shown in A.2.1.12.

Table 12—Parameter list for creating dense tensor with a sequence of number

Parameter	Type	Optional/Required	Description
Start	Input	Required	The start of the interval.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer and float; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
End	Input	Optional	The end of the interval; default can be none.
Step	Input	Optional	The spacing between values; default can be 1.

7.1.12.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.13 Create dense tensor in a linspace

7.1.13.1 Function

Create a dense tensor with a fixed number of evenly spaced values within a given interval l.

7.1.13.2 Interface parameters

The interface shall comply with Table 13 and the C code example (op_linspace) is see A.2.1.13.

Table 13—Parameter list for creating dense tensor in a linspace

Parameter	Type	Optional/Required	Description
Start	Input	Required	The start of the interval.
Stop	Input	Required	The end of the interval.
Number of data	Input	Required	The number of elements.
Output tensor	Output	Required	Newly created tensor.
Data type	Input	Optional	The data type can be integer and float; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.

7.1.13.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The interval specified by the starting point, endpoint, and number of data is illegal.

Out of memory: Insufficient space allocated for creating tensor.

7.1.14 Create sparse tensor

7.1.14.1 Function

Create a sparse tensor.

7.1.14.2 Interface parameters

The interface shall conform to Table 14, and the C code example (op_create_sparse_tensor) is shown in A.2.1.14.

Table 14—Parameter list for creating a sparse tensor

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Indices	Input	Required	The coordinates of the nonzero element in the sparse tensor; shape is [NDIM, NNZ].
Value	Input	Required	The nonzero values; the shape is [NNZ].
Number of nonzero elements	Input	Required	The number of nonzero elements (NNZ).
Output tensor	Output	Required	Newly created tensor
Data type	Input	Optional	The data type can be integer, float, complex and boolean, etc.; default can be float.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.

7.1.14.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: An internal call operation error occurred.

7.1.15 Create quantitative tensor

7.1.15.1 Function

Create a quantitative tensor. If there is an initialization array, initialize it with its value; otherwise, create an empty tensor.

7.1.15.2 Interface parameters

The interface shall conform to Table 15, and the C code example (op_create_quant_tensor) is shown in A.2.1.15.

Table 15—Parameter list for creating quantitative tensor

Parameter	Type	Optional/Required	Description
Tensor shape	Input	Required	An array of dimension sizes.
Initialization array	Input	Required	The array to initialize the tensor data; if empty, then a tensor full of zero is created; default can be empty.
Length	Input	Required	The amount of data to be copied, measured in bytes. The insufficient part shall be supplemented with 0; default can be 0.
Output tensor	Output	Required	Newly created quantitative tensor.
Data type	Input	Optional	The data type can be QT_SYM_INT8, QT_SYM_INT16, QT_SYM_PER_CHANNEL_INT8, QT_SYM_PER_CHANNEL_INT16, QT_ASYM_UINT8, QT_ASYM_INT8, QT_ASYM_UINT16, and QT_ASYM_INT16; default can be QT_SYM_INT8.
Device type	Input	Optional	The device type can be CPU, GPU, etc.; default can be CPU.
Scaling factor	Input	Optional	The scaling factor.
Zero point	Input	Optional	The size of movement in the horizontal direction.

7.1.15.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Dimension mismatch: The dimensions of the tensor and the initialization array do not match.

Other internal errors: Internal call operation error occurred.

7.1.15.4 Other

Tensor-wise quantization: All data inside a tensor share the same scale. Supported data types are 8-bit signed symmetric quantization integer and 16-bit signed symmetric quantization integer, optional support for 8-bit unsigned asymmetric quantization integer, 8-bit signed asymmetric quantization integer, 16-bit unsigned asymmetric quantization integer, and 16-bit signed asymmetric quantized integer.

Channel-wise quantization: The data of the same channel inside the tensor share the same scale. The supported data types are 8-bit signed channel-level symmetric quantized integer and 16-bit signed channel-level symmetric quantized integer.

Asymmetric quantization should conform to Equation (1) to Equation (3).

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

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

$$q = \frac{r - a}{s} \quad (3)$$

where x is the floating-point value to be quantized, $[a, b]$ is the quantization range, a is the minimum value of the float to be quantized, and b is the maximum value of the float to be quantized. The result was rounded to the nearest whole number. If the quantization level is k , then n is $2 \times k$. For example, if k is 8, then n is 256. q is the quantized integer.

Symmetric quantization should conform to Equation (4) and Equation (5).

$$M = \max(\text{abs}(x)) \quad (4)$$

$$q = \frac{x}{M} \times (n - 1) \quad (5)$$

where x is the floating-point value to be quantized, and M is the absolute maximum value in the float to be quantized. For 8-bit quantization, $n = 128$. q is the quantized integer. Whether it is symmetric quantization or asymmetric quantization, it should conform to Equation (6).

$$x = (q - \text{zero_point}) \times \text{scale} \quad (6)$$

The symmetric quantization method is a special case of the asymmetric method; that is, $\text{zero_point} = 0$.

7.1.16 Copy tensor

7.1.16.1 Function

Copy a tensor.

7.1.16.2 Interface parameters

The interface shall conform to Table 16, and the C code example (op_duplicate) is shown in A.2.1.16.

Table 16—Parameter list for copying tensor

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor being copied.
Output tensor	Output	Required	The output tensor.

7.1.16.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: Internal call operation error occurred.

7.1.16.4 Others

The data of the new tensor is the same as the input tensor but stored in a newly allocated space.

7.1.17 Copy diagonal element

7.1.17.1 Function

Create a 2-D square tensor with the elements of the 1-D input tensor as the diagonal.

7.1.17.2 Interface parameters

The interface shall conform to Table 17, and the C code example (op_diag) is shown in A.2.1.17.

Table 17—Parameter list for copying diagonal elements

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor; the shape is [N].
Output tensor	Output	Required	The output tensor; the shape is [N, N].

7.1.17.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: Internal call operation error occurred.

7.1.18 Destroy tensor

7.1.18.1 Function

Destroy a specified tensor.

7.1.18.2 Interface parameters

The interface shall conform to Table 18, and the C code example (op_destroy) is shown in A.2.1.18.

Table 18—Parameter list for destroying tensor

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be destroyed.

7.1.18.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

Object not initialized: A null pointer was passed and could not be destroyed.

Other internal errors: Internal call operation error occurred.

7.2 Query and inspection

7.2.1 Shape query

7.2.1.1 Function

Get the shape of the tensor.

7.2.1.2 Interface parameters

The interface shall conform to Table 19, and the C code example (op_shape) is shown in A.2.2.1.

Table 19—Parameter list for the shape query

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be queried shape.
Output tensor	Output	Required	The output tensor, data type is integer; the shape is [NDIM].

7.2.1.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.2 Finite inspection

7.2.2.1 Function

Check whether every element of the input tensor is finite number; return true for finite number.

7.2.2.2 Interface parameters

The interface shall conform to Table 20, and the C code example (op_isfinite) is shown in A.2.2.2.

Table 20—Parameter list for finite inspection

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be inspected.
Output tensor	Output	Required	The result tensor; the shape is the same as input, and the data type is boolean.

7.2.2.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.3 Infinite inspection

7.2.3.1 Function

Check whether every element of the input tensor is finite number; return true for infinite number.

7.2.3.2 Interface parameters

The interface shall conform to Table 21, and the C code example (op_has_inf) is shown in A.2.2.3.

Table 21—Parameter list for infinite inspection

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be inspected.
Output tensor	Output	Required	The result tensor; the shape is the same as input, and the data type is boolean.

7.2.3.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.4 NaN inspection

7.2.4.1 Function

Check whether every element of the input tensor is Not a Number (NaN); return true for NaN.

7.2.4.2 Interface parameters

The interface shall conform to Table 22, and the C code example (op_has_nan) is shown in A.2.2.4.

Table 22—Parameter list for NaN inspection function

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be inspected.
Output tensor	Output	Required	The result tensor; the shape is the same as input, and the data type is boolean.

7.2.4.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.5 Number of elements query

7.2.5.1 Function

Get the total number of elements in the input tensor.

7.2.5.2 Interface parameters

The interface shall conform to Table 23, and the C code example (op_size) is shown in A.2.2.5.

Table 23—Parameter list for number of elements query

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be queried
Output tensor	Output	Required	The output tensor, shape is [1]

7.2.5.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.6 Rank query

7.2.6.1 Function

Get the rank of the input tensor.

7.2.6.2 Interface parameters

The interface shall conform to Table 24, and the C code example (op_rank) is shown in A.2.2.6.

Table 24—Parameter list for rank query

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be queried.
Output tensor	Output	Required	The tensor of the input rank; the shape is [1].

7.2.6.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The parameters were wrong.

7.2.7 Contiguous memory inspection

7.2.7.1 Function

Check whether the data of the input tensor is continuously stored in the memory; if it is, return true.

7.2.7.2 Interface parameters

The interface shall conform to Table 25, and the C code example (op_is_contiguous) is shown in A.2.2.7.

Table 25—Parameter list for continuous memory inspection

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be inspected.
Output tensor	Output	Required	The result tensor; the shape is [1], and the data type is boolean.

7.2.7.3 Interface returns

No errors: The operation was successful.

Object not initialized: The input tensor object is illegal.

Illegal parameters: Indicates that other parameters are illegal.

Other internal errors: Internal call operation error occurred.

7.3 Tensor conversion

7.3.1 Cast

7.3.1.1 Function

Cast the input tensor to an output tensor with the new specified data type.

7.3.1.2 Interface parameters

The interface shall conform to Table 26, and the C code example (op_cast) is shown in A.2.3.1.

Table 26—Parameter list for casting

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be casted.
Data type	Input	Required	The data type to be casted to.
Output tensor	Output	Required	The tensor after casting.

7.3.1.3 Interface returns

No errors: The operation was successful.

Type mismatch: The type of the input tensor object is incompatible with the type to be converted.

Object not initialized: The input tensor object is illegal

Out of memory: Insufficient space was allocated to create a tensor.

Other internal errors: Internal call operation error occurred.

7.3.2 Reshape

7.3.2.1 Function

Change the shape of the input tensor without changing its data.

7.3.2.2 Interface parameters

The interface shall conform to Table 27, and the C code example (op_reshape) is shown in A.2.3.2.

Table 27—Parameter list for reshaping

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor to be reshaped.
Shape	Input	Required	The target shape.
Output tensor	Output	Required	The tensor after reshaping.

7.3.2.3 Interface returns

No errors: The operation was successful.

Dimension mismatch: The total size of the input tensor and the output tensor are inconsistent.

Object not initialized: The input tensor object is illegal.

Out of memory: Insufficient space was allocated to create a tensor.

Illegal parameters: Indicates that other parameters are illegal.

Other internal errors: An internal call operation error occurred.

7.3.3 Expand dimension

7.3.3.1 Function

Add an additional dimension to the input tensor at the given axis.

7.3.3.2 Interface parameters

The interface shall conform to Table 28, and the C code example (op_expand_dims) is shown in A.2.3.3.

Table 28—Parameter list for expanding dimension

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be expanded.
Axis	Input	Required	The position where to add a new dimension.
Output tensor	Output	Required	The tensor after inserting the new dimension.

7.3.3.3 Interface returns

No errors: The operation was successful.

Out of range: The insertion position exceeded the dimension of the input tensor.

Object not initialized: The input tensor object was illegal.

Out of memory: Insufficient space was allocated to create a tensor.

Illegal parameters: Indicates that other parameters were illegal.

Other internal errors: Internal call operation error occurred.

7.3.4 Squeeze dimensions

7.3.4.1 Function

Squeeze the dimensions of size 1 of the input tensor's shape.

7.3.4.2 Interface parameters

The interface shall conform to Table 29, and the C code example (op_squeeze) is shown in A.2.3.4.

Table 29—Parameter list for squeezing dimensions

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be squeezed.
Axes	Input	Required	The array of positions where the dimension is to be squeezed.
Output tensor	Output	Required	The tensor after squeezing.

7.3.4.3 Interface returns

No errors: The operation was successful.

Out of range: The squeezed position exceeded the dimension of the input tensor.

Object not initialized: The input tensor object was illegal.

Out of memory: Insufficient space was allocated to create a tensor.

Illegal parameters: Indicates that other parameters were illegal.

Other internal errors: Internal call operation error occurred.

7.3.5 Transpose

7.3.5.1 Function

Permute the dimensions of the input tensor according to input permutation.

7.3.5.2 Interface parameters

The interface shall conform to Table 30, and the C code example (op_transpose) is shown in A.2.3.5.

Table 30—Parameter list for transposing

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be transposed.
Permutation	Input	Required	An array of permutation dimensions.
Output tensor	Output	Required	The tensor after transposition.

7.3.5.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: Internal call operation error occurred.

7.3.6 Split

7.3.6.1 Function

Split the input tensor into multiple tensors along the given axis.

7.3.6.2 Interface parameters

The interface shall conform to Table 31, and the C code example (op_split) is shown in A.2.3.6.

Table 31—Parameter list for tensor split

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be split.
Split size array	Input	Required	The array of split size.
Output tensor array	Output	Required	The result array of tensor.
Axis	Input	Optional	The dimension along which to split; default can be 0.

7.3.6.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: Internal call operation error occurred.

7.3.7 Concat

7.3.7.1 Function

Concatenates the input tensors along the axis.

7.3.7.2 Interface parameters

The interface shall conform to Table 32, and the C code example (op_concat) is shown in A.2.3.7.

Table 32—Parameter list for concatenation

Parameter	Type	Optional/Required	Description
Input tensor array	Input	Required	The array of the input tensors to be concatenated.
Output tensor	Output	Required	The concatenated tensor.
Axis	Input	Optional	The dimension along which to concatenate; default can be 0.

7.3.7.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal call operation error occurred.

7.3.8 Stack

7.3.8.1 Function

Stacks all the input tensors along the axis dimension. All tensors must be of the same shape and same data type.

7.3.8.2 Interface parameters

The interface shall conform to Table 33, and the C code example (op_stack) is shown in A.2.3.8.

Table 33—Parameter list for stacking

Parameter	Type	Optional/Required	Description
Input tensor array	Input	Required	The input tensor array to be stacked.
Output tensor	Output	Required	The stacked tensor.
Axis	Input	Optional	The dimension along which to stack; default can be 0.

7.3.8.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal call operation error occurred.

7.3.9 Unstack

7.3.9.1 Function

Unstack the input tensor into several tensors along the axis.

7.3.9.2 Interface parameters

The interface shall conform to Table 34, and the C code example (op_unstack) is shown in A.2.3.9.

Table 34—Parameter list for unstacking

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor to be unstacked.
Output tensor array	Output	Required	The output tensor array after unstacking.
Axis	Input	Optional	The specified dimension to unstack; default can be 0.

7.3.9.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal call operation error occurred.

7.3.10 Slice

7.3.10.1 Function

Slice the input tensor into the specified shape.

7.3.10.2 Forward interface parameters

The interface shall conform to Table 35, and the C code example (op_slice) is shown in A.2.3.10.

Table 35—Parameter list for forward interface of slicing

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The input tensor to slice.
Starts	Input	Required	The starting indices of corresponding axis in axes.
Ends	Input	Required	The ending indices of corresponding axis in axes.
Axes	Input	Required	The axes to apply slicing.
Output tensor	Output	Required	The tensor after slicing.

7.3.10.3 Forward Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal call operation error occurred.

7.3.10.4 Backward interface parameters

The interface shall conform to Table 36, and the C code example (op_slice_backward) is shown in A.2.3.10.

Table 36—Parameter list for backward interface of slicing

Parameter	Type	Optional/Required	Description
Output gradient	Input	Required	The gradient tensor of the output tensor.
Input tensor	Input	Required	The input tensor.

Parameter	Type	Optional/Required	Description
Starts	Input	Required	The starting indices of corresponding axis in axes.
Ends	Input	Required	The ending indices of corresponding axis in axes.
Axes	Input	Required	The axes to apply slicing.
Input gradient	Output	Required	The gradient tensor of the input tensor.

7.3.10.5 Backward interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: Internal call operation error occurred.

7.3.11 Tile

7.3.11.1 Function

Construct a tensor by repeating the elements of the input tensor.

7.3.11.2 Interface parameters

The interface shall conform to Table 37, and the C code example (op_tile) is shown in A.2.3.11.

Table 37—Parameter list for tile

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The source tensor.
Repetition	Input	Required	The number of repetitions of each dimension.
Output tensor	Output	Required	The tensor after tile.

7.3.11.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: Internal call operation error occurred.

7.3.12 Pad

7.3.12.1 Function

Pad the input tensor according to the paddings.

7.3.12.2 Interface parameters

The interface shall conform to Table 38, and the C code example (op_pad) is shown in A.2.3.12.

Table 38—Parameter list for padding

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be padded
Padding array	Input	Required	The padding size of each dimension. The size should be 2*NDIM.
Value	Input	Required	The value to fill the padded areas; default can be 0.
Mode	Input	Required	Can be constant, reflect, replicate, or circular; default can be constant.
Output tensor	Output	Required	The tensor after padding.

7.3.12.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object is illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters are illegal.
Other internal errors: Internal call operation error occurred.

7.3.13 Reverse

7.3.13.1 Function

Reverse specific dimensions of the input tensor.

7.3.13.2 Interface parameters

The interface shall conform to Table 39, and the C code example (op_reverse) is shown in A.2.3.13.

Table 39—Parameter list for reversing

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be reversed.
Output tensor	Output	Required	The tensor after reversing.
Axes	Input	Optional	The axes to apply reversing; default can be 0.

7.3.13.3 Interface returns

No errors: The operation was successful.
Out of range: The reversed positions exceeded the dimension of the input tensor.
Object not initialized: The input tensor object was illegal.
Out of memory: Insufficient space was allocated to create a tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal call operation error occurred.

7.3.14 Roll

7.3.14.1 Function

Roll the input tensor along the given axis or axes.

7.3.14.2 Interface parameters

The interface shall conform to Table 40, and the C code example (op_roll) is shown in A.2.3.14.

Table 40—Parameter list for rolling

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be rolled.
Shift	Input	Required	The number of places by which elements are shifted positively (toward larger indices) along the specified axis.
Axes	Input	Required	The axes to apply rolling.
Output tensor	Output	Required	The tensor after rolling.

7.3.14.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The assigned axis exceeded the dimension of the input tensor.

7.3.15 Clip

7.3.15.1 Function

Clip all elements in the input tensor into the range [min, max]. Values outside the range are clipped to the range edges.

7.3.15.2 Forward interface parameters

The interface shall conform to Table 41, and the C code example (op_clip) is shown in A.2.3.15.

Table 41—Parameter list for forward interface of clipping

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be clipped.
Lower bound	Input	Required	The lower bound of the value range.
Upper bound	Input	Required	The upper bound of the value range.
Output tensor	Output	Required	The tensor after clipping.

7.3.15.3 Forward interface returns

No errors: The operation was successful.

Object not initialized: The input tensor object was invalid.

Illegal parameters: Indicates that other parameters were illegal.

7.3.15.4 Backward interface parameters

The interface shall conform to Table 42, and the C code example (op_clip_backward) is shown in A.2.3.15.

Table 42—Parameter list for backward interface of clipping

Parameter	Type	Optional/Required	Description
Output gradient	Input	Required	The gradient of the output tensor.
Input gradient	Output	Required	The gradient tensor of the input tensor.
Lower bound	Input	Optional	The lower bound of the value range.
Upper bound	Input	Optional	The upper bound of the value range.

7.3.15.5 backward interface returns

No errors: The operation was successful.

Object not initialized: The input tensor object was invalid.

Illegal parameters: Indicated that other parameters were illegal.

7.3.16 Gather

7.3.16.1 Function

Gather values along an axis specified by the index.

7.3.16.2 Interface parameters

The interface shall conform to Table 43, and the C code example (op_gather) is shown in A.2.3.16.

Table 43—Parameter list for gathering

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The source tensor.
Index tensor	Input	Required	The indices of elements to gather.
Axis	Input	Required	The axis to apply gathering.
Output tensor	Output	Required	The destination tensor.

7.3.16.3 Interface returns

No errors: The operation was successful.

Object not initialized: The input tensor object was invalid.

Illegal parameters: Indicated that other parameters were illegal.

7.3.17 Scatter

7.3.17.1 Function

Update the input tensor on selected indices based on the corresponding updating data.

7.3.17.2 Interface parameters

The interface shall conform to Table 44, and the C code example (op_scatter) is shown A.2.3.17.

Table 44—Parameter list for scattering

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be updated; the shape can be [N, *].
Index tensor	Input	Required	The indices of elements to scatter; the shape can be [M], the value cannot exceed N, and the data type can be integer.
Update tensor	Input	Required	The corresponding updates data; the shape can be [M, *].
Output tensor	Output	Required	The updated tensor with the same shape and data type with the input tensor.
Overwrite	Input	Optional	Whether to overwrite the input when updating; default can be true.

7.3.17.3 Interface returns

No errors: The operation was successful.

Object not initialized: The input tensor object was invalid.

Illegal parameters: Indicated that other parameters were illegal.

7.3.18 Flatten

7.3.18.1 Function

Flatten a contiguous range of axes in the input tensor according to the start axis and stop axis.

7.3.18.2 Interface parameters

The interface shall conform to Table 45, and the C code example (op_flatten) is shown in A.2.3.18.

Table 45—Parameter list for flattening

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be flattened.
Output tensor	Output	Required	The tensor after flattening.
Start axis	Input	Optional	The first axis to flatten from.
End axis	Input	Optional	The last axis to flatten to.

7.3.18.3 Interface returns

No errors: The operation was successful.
Dimension error: Invalid starting or ending dimension.
Object not initialized: Invalid input tensor.
Out of memory: Lack of enough memory allocated for output tensor.
Illegal parameters: Indicates that other parameters were illegal.
Other internal errors: Internal invocation error.

7.3.19 Flip

7.3.19.1 Function

Flip the tensor along the specified axis.

7.3.19.2 Interface parameters

The interface shall conform to Table 46, and the C code example (op_flip) is shown in A.2.3.19.

Table 46—Parameter list for flipping

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to be flipped.
Axis	Input	Required	Axis to apply flipping.
Output tensor	Output	Required	The tensor after flipping.

7.3.19.3 Interface returns

No errors: The operation was successful.
Axis error: The axis was illegal.
Object not initialized: The input tensor was invalid.
Out of memory: Not enough memory allocated for output tensor.
Illegal parameters: The other parameters were illegal.
Other internal errors: Internal invocation error.

7.3.20 Sign

7.3.20.1 Function

Perform the sign of the input tensor element-wise, 1 for positive, −1 for negative, and 0 for zero element.

7.3.20.2 Interface parameters

The interface shall conform to Table 47, and the C code example (op_sign) is shown in A.2.3.20.

Table 47—Parameter list for sign

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	The tensor to perform sign of.
Output tensor	Output	Required	The result tensor.

7.3.20.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was invalid.
Out of memory: Not enough memory allocated for output tensor.
Illegal parameters: The other parameters were illegal.
Other internal errors: Internal invocation error.

7.3.21 Where

7.3.21.1 Function

Select element from x or y based on the corresponding condition, as is shown in Equation (7).

$$\text{out}_i = \begin{cases} x_i, & \text{if } \text{cond}_i \text{ is true} \\ y_i, & \text{if } \text{cond}_i \text{ is false} \end{cases} \quad (7)$$

7.3.21.2 Interface parameters

The interface shall conform to Table 48, and the C code example (op_where) is shown in A.2.3.21.

Table 48—Parameter list for where

Parameter	Type	Optional/Required	Description
Input condition	Input	Required	The condition to choose x or y ; data type can boolean.
Input tensor 1	Input	Required	The 42irst tensor, x in equation.
Input tensor 2	Input	Required	The second tensor, y in equation.
Output tensor	Output	Required	The result tensor.

7.3.21.3 Interface returns

No errors: The operation was successful.
Object not initialized: The input tensor object was invalid.
Out of memory: Lack of enough memory allocated for output tensor.
Illegal parameters: The other parameters were illegal.
Other internal errors: An error occurred in the internal call operation.

7.4 Arithmetic operations

7.4.1 Add

7.4.1.1 Function

Add two input tensors element-wise, as is shown in Equation (8).

$$z_i = x_i + y_i \quad (8)$$

7.4.1.2 Interface parameters

The interface shall conform to Table 49, and the C code example (op_add) is shown in A.2.4.1.

Table 49—Parameter list for adding

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.1.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of parameters did not match.

7.4.2 Sub

7.4.2.1 Function

Subtract the second input tensor from the first input tensor element-wise, as is shown in Equation (9).

$$z_i = x_i - y_i \quad (9)$$

7.4.2.2 Interface parameters

The interface shall conform to Table 50, and the C code example (op_sub) is shown in A.2.4.2.

Table 50—Parameter list for subtraction

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.2.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of parameters did not match.

7.4.3 Multiply

7.4.3.1 Function

Multiply two tensors element-wise, as is shown in Equation (10).

$$z_i = x_i \times y_i \quad (10)$$

7.4.3.2 Interface parameters

The interface shall conform to Table 51, and the C code example (op_mul) is shown in A.2.4.3.

Table 51—Parameter list for multiplying

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.3.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of parameters did not match.

7.4.4 Multiply and add

7.4.4.1 Function

Multiply two tensors element-wise, and add with the third tensor element-wise, as is shown in Equation (11).

$$z_i = x_i \times y_i + a_i \quad (11)$$

7.4.4.2 Interface parameters

The interface shall conform to Table 52, and the C code example (op_muladd) is shown in A.2.4.4.

Table 52—Parameter list for multiplying and adding

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be $[*]$, data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Input tensor 3	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, a in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.4.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of parameters did not match.

7.4.5 Divide

7.4.5.1 Function

Divide two tensors element-wise, as is shown in Equation (12).

$$z_i = \frac{x_i}{y_i} \quad (12)$$

7.4.5.2 Interface parameters

The interface shall conform to Table 53, and the C code example (op_div) is shown in A.2.4.5.

Table 53—Parameter list for dividing

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	The dividend; the shape can be $[*]$, data type can be integer or float, x in equation.
Input tensor 2	Input	Required	The divisor; the shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.5.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.6 Floor divide

7.4.6.1 Function

Divide and round down two tensors element-wise, as is shown in Equation (13).

$$z_i = \left[\frac{x_i}{y_i} \right] \quad (13)$$

7.4.6.2 Interface parameters

The interface shall conform to Table 54, and the C code example (op_floordiv) is shown in A.2.4.6.

Table 54—Parameter list for floor dividing

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	The dividend; the shape can be $[*]$, data type can be integer or float, x in equation.
Input tensor 2	Input	Required	The divisor; the shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.6.3 Interface returns

No errors: The operation was successful.

Type mismatch: The data types of parameters did not match.

7.4.7 Mod

7.4.7.1 Function

Compute the remainder by dividing the first input tensor by the second input tensor element-wise, as is shown in Equation (14).

$$z_i = x_i \% y_i \quad (14)$$

7.4.7.2 Interface parameters

The interface shall conform to Table 55, and the C code example (op_mod) is shown in A.2.4.7.

Table 55—Parameter list for mod

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be $[*]$, data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.7.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.8 Maximum

7.4.8.1 Function

Compute the maximum of input tensors element-wise, as is shown in Equation (15).

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

7.4.8.2 Interface parameters

The interface shall conform to Table 56, and the C code example (op_max) is shown in A.2.4.8.

Table 56—Parameter list for maximum

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.8.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.9 Minimum

7.4.9.1 Function

Compute the minimum of input tensors element-wise, as is shown in Equation (16).

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

7.4.9.2 Interface parameters

The interface shall conform to Table 57, and the C code example (op_min) is shown in A.2.4.9.

Table 57—Parameter list for minimum

Parameter	Type	Optional/Required	Description
Input tensor 1	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Input tensor 2	Input	Required	Shape should be the same as input tensor 1 or broadcastable with it, data type should be the same as input tensor 1, y in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as the one after broadcasting, data type should be the same as input tensor 1, z in equation.

7.4.9.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.10 Abs

7.4.10.1 Function

Compute the absolute value of the input tensor element-wise, as is shown in Equation (17).

$$y_i = |x_i| \quad (17)$$

7.4.10.2 Interface parameters

The interface shall conform to Table 58, and the C code example (op_abs) is shown in A.2.4.10.

Table 58—Parameter list for abs

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as input, data type should be the same as input tensor 1, y in equation.

7.4.10.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.11 Reciprocal

7.4.11.1 Function

Compute the reciprocal of the input tensor element-wise, as is shown in Equation (18).

$$y_i = \frac{1}{x_i} \quad (18)$$

7.4.11.2 Interface parameters

The interface shall conform to Table 59, and the C code example (op_reciprocal) is shown in A.2.4.11.

Table 59—Parameter list for reciprocal

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	Shape can be [*], data type can be integer or float, x in equation.
Output tensor	Output	Required	The result tensor; the shape should be the same as input, data type should be the same as input tensor 1, y in equation.

7.4.11.3 Interface returns

No errors: The operation was successful.

Illegal parameters: The input parameters were invalid.

Type mismatch: The data types of the parameters did not match.

7.4.12 Trace

7.4.12.1 Function

Compute the sum along the diagonals of the input tensor.

7.4.12.2 Forward interface parameters

The interface shall conform to Table 60, and the C code example (op_trace) is shown in A.2.4.12.

Table 60—Parameter list for forward interface trace

Parameter	Type	Optional/Required	Description
Input tensor	Input	Required	Shape can be [N, D, *]; data type can be integer or float.
Output tensor	Output	Required	The result tensor.
Offset	Input	Optional	Which diagonals in input tensor will be taken; default can be 0.
Axis 1	Input	Optional	The first axis with respect to take diagonal; default can be 0.
Axis 2	Input	Optional	The second axis with respect to take diagonal; default can be 1.

7.4.12.3 Forward interface returns

No errors: The operation was successful.

Illegal parameters: The input tensor's dimension was less than 2.

7.4.12.4 Backward interface parameters

The interface shall conform to Table 61, and the C code example (op_trace_backward) is shown in A.2.4.12.

Table 61—Parameter list for backward interface of trace

Parameter	Type	Optional/Required	Description
Output gradient	Input	Required	The gradient of output tensor.
Input tensor	Output	Required	The gradient of input tensor.
Offset	Input	Optional	Which diagonals in input tensor will be taken; default can be 0.
Axis 1	Input	Optional	The first axis with respect to take diagonal; default can be 0.
Axis 2	Input	Optional	The second axis with respect to take diagonal; default can be 1.

7.4.12.5 Backward interface returns

No errors: The operation was successful.

Illegal parameters: The input tensor's dimension was less than 2.

7.5 Comparison operation

7.5.1 Equal

7.5.1.1 Function

Computes the equivalence between two tensors element-wise, as is shown in Equation (19).