torch

The torch package contains ata structures for multi- imensional tensors an efines mathematical operations over these tensors. A itionally, it provi es many utilities for efficient serialization of Tensors an ar itrary types, an other useful utilities.

It has a C DA counterpart, that ena les you to run your tensor computations on an NVIDIA GP with compute capa ility >= 3.0.

Tensors

`is_tensor`	Returns True if *obj* is a PyTorch tensor.
`is_storage`	Returns True if *obj* is a PyTorch storage o ect.
`is_complex`	Returns True if the ata type of `input` is a complex ata type i.e., one of `torch.complex64`, an `torch.complex128`.
`is_conj`	Returns True if the `input` is a con ugate tensor, i.e. its con ugate it is set to *True*.
`is_floating_point`	Returns True if the ata type of `input` is a floating point ata type i.e., one of `torch.float64`, `torch.float32`, `torch.float16`, an `torch.bfloat16`.
`is_nonzero`	Returns True if the `input` is a single element tensor which is not equal to zero after type conversions.
`set_default_dtype`	Sets the efault floating point type to `d`.
`get_default_dtype`	Get the current efault floating point `torch.dtype`.
`set_default_device`	Sets the efault `torch.Tensor` to e allocate on `device`.
`get_default_device`	Gets the efault `torch.Tensor` to e allocate on `device`
`set_default_tensor_type`	
`numel`	Returns the total num er of elements in the `input` tensor.
`set_printoptions`	Set options for printing.
`set_flush_denormal`	Disa les enormal floating num ers on CP .

Creation O s

• NOTE

Ran om sampling creation ops are liste un er Ran om sampling an inclu e: `torch.rand()` `torch.rand_like()` `torch.randn()` `torch.randn_like()` `torch.randint()` `torch.randint_like()` `torch.randperm()` You may also use `torch.empty()` with the In-place ran om sampling metho s to create `torch.Tensor` s with values sample from a roa er range of istri utions.

`tensor`	Constructs a tensor with no autogra history (also known as a "leaf tensor", see Autogra mechanics) y copying `data`.
`sparse_coo_tensor`	Constructs a sparse tensor in COO(r inate) format with specifie values at the given `indices`.
`sparse_csr_tensor`	Constructs a sparse tensor in CSR (Compresse Sparse Row) with specifie values at the given `crow_indices` an `col_indices`.
`sparse_csc_tensor`	Constructs a sparse tensor in CSC (Compresse Sparse Column) with specifie values at the given `ccol_indices` an `row_indices`.
`sparse_bsr_tensor`	Constructs a sparse tensor in BSR (Block Compresse Sparse Row)) with specifie 2- imensional locks at the given `crow_indices` an `col_indices`.
`sparse_bsc_tensor`	Constructs a sparse tensor in BSC (Block Compresse Sparse Column)) with specifie 2- imensional locks at the given `ccol_indices` an `row_indices`.
`asarray`	Converts `obj` to a tensor.

`as_strided`	`size`, `stride` an `storage_offset`.
`from_file`	Creates a CP tensor with a storage acke y a memory-mappe file.
`from_numpy`	Creates a `Tensor` from a `numpy.ndarray`.
`from_dlpack`	Converts a tensor from an external li rary into a `torch.Tensor`.
`frombuffer`	Creates a 1- imensional `Tensor` from an o ect that implements the Python uffer protocol.
`zeros`	Returns a tensor fille with the scalar value *0*, with the shape efine y the varia le argument `size`.
`zeros_like`	Returns a tensor fille with the scalar value *0*, with the same size as `input`.
`ones`	Returns a tensor fille with the scalar value *1*, with the shape efine y the varia le argument `size`.
`ones_like`	Returns a tensor fille with the scalar value *1*, with the same size as `input`.
`arange`	Returns a 1-D tensor of size $\left\lfloor \frac{\text{end}-\text{start}}{\text{step}} \right\rfloor$ with values from the interval `[start, end)` taken with common ifference `step` eginning from *start*.
`range`	Returns a 1-D tensor of size $\left\lfloor \frac{\text{end}-\text{start}}{\text{step}} \right\rfloor + 1$ with values from `start` to `end` with step `step`.
`linspace`	Creates a one- imensional tensor of size `steps` whose values are evenly space from `start` to `end`, inclusive.
`logspace`	Creates a one- imensional tensor of size `steps` whose values are evenly space from $\text{base}^{\text{start}}$ to base^{end}, inclusive, on a logarithmic scale with ase `base`.
`eye`	Returns a 2-D tensor with ones on the iagonal an zeros elsewhere.
`empty`	Returns a tensor fille with uninitialize ata.
`empty_like`	Returns an uninitialize tensor with the same size as `input`.
`empty_strided`	Creates a tensor with the specifie `size` an `stride` an fille with un efine ata.
`full`	Creates a tensor of size `size` fille with `fill_value`.
`full_like`	Returns a tensor with the same size as `input` fille with `fill_value`.
`quantize_per_tensor`	Converts a float tensor to a quantize tensor with given scale an zero point.
`quantize_per_channel`	Converts a float tensor to a per-channel quantize tensor with given scales an zero points.
`dequantize`	Returns an fp32 Tensor y equantizing a quantize Tensor
`complex`	Constructs a complex tensor with its real part equal to `real` an its imaginary part equal to `imag`.
`polar`	Constructs a complex tensor whose elements are Cartesian coor inates correspon ing to the polar coor inates with a solute value `abs` an angle `angle`.
`heaviside`	Computes the Heavisi e step function for each element in `input`.

In exing, Slicing, Joining, Mutating O s

`adjoint`	Returns a view of the tensor con ugate an with the last two imensions transpose .
`argwhere`	Returns a tensor containing the in ices of all non-zero elements of `input`.
	Concatenates the given sequence of tensors in `tensors` in the

`concat`	
`concatenate`	Alias of `torch.cat()`.
`conj`	Returns a view of `input` with a flippe con ugate it.
`chunk`	Attempts to split a tensor into the specifie num er of chunks.
`dsplit`	Splits `input`, a tensor with three or more imensions, into multiple tensors epthwise accor ing to `indices_or_sections`.
`column_stack`	Creates a new tensor y horizontally stacking the tensors in `tensors`.
`dstack`	Stack tensors in sequence epthwise (along thir axis).
`gather`	Gathers values along an axis specifie y *dim*.
`hsplit`	Splits `input`, a tensor with one or more imensions, into multiple tensors horizontally accor ing to `indices_or_sections`.
`hstack`	Stack tensors in sequence horizontally (column wise).
`index_add`	See `index_add_()` for function escription.
`index_copy`	See `index_add_()` for function escription.
`index_reduce`	See `index_reduce_()` for function escription.
`index_select`	Returns a new tensor which in exes the `input` tensor along imension `dim` using the entries in `index` which is a *LongTensor*.
`masked_select`	Returns a new 1-D tensor which in exes the `input` tensor accor ing to the oolean mask `mask` which is a *BoolTensor*.
`movedim`	Moves the imension(s) of `input` at the position(s) in `source` to the position(s) in `destination`.
`moveaxis`	Alias for `torch.movedim()`.
`narrow`	Returns a new tensor that is a narrowe version of `input` tensor.
`narrow_copy`	Same as `Tensor.narrow()` except this returns a copy rather than share storage.
`nonzero`	
`permute`	Returns a view of the original tensor `input` with its imensions permute .
`reshape`	Returns a tensor with the same ata an num er of elements as `input`, ut with the specifie shape.
`row_stack`	Alias of `torch.vstack()`.
`select`	Slices the `input` tensor along the selecte imension at the given in ex.
`scatter`	Out-of-place version of `torch.Tensor.scatter_()`
`diagonal_scatter`	Em e s the values of the `src` tensor into `input` along the iagonal elements of `input`, with respect to `dim1` an `dim2`.
`select_scatter`	Em e s the values of the `src` tensor into `input` at the given in ex.
`slice_scatter`	Em e s the values of the `src` tensor into `input` at the given imension.
`scatter_add`	Out-of-place version of `torch.Tensor.scatter_add_()`
`scatter_reduce`	Out-of-place version of `torch.Tensor.scatter_reduce_()`

`squeeze`	Returns a tensor with all specifie imensions of `input` of size *1* remove .
`stack`	Concatenates a sequence of tensors along a new imension.
`swapaxes`	Alias for `torch.transpose()`.
`swapdims`	Alias for `torch.transpose()`.
`t`	Expects `input` to e <= 2-D tensor an transposes imensions 0 an 1.
`take`	Returns a new tensor with the elements of `input` at the given in ices.
`take_along_dim`	Selects values from `input` at the 1- imensional in ices from `indices` along the given `dim`.
`tensor_split`	Splits a tensor into multiple su -tensors, all of which are views of `input`, along imension `dim` accor ing to the in ices or num er of sections specifie y `indices_or_sections`.
`tile`	Constructs a tensor y repeating the elements of `input`.
`transpose`	Returns a tensor that is a transpose version of `input`.
`unbind`	Removes a tensor imension.
`unravel_index`	Converts a tensor of flat in ices into a tuple of coor inate tensors that in ex into an ar itrary tensor of the specifie shape.
`unsqueeze`	Returns a new tensor with a imension of size one inserte at the specifie position.
`vsplit`	Splits `input`, a tensor with two or more imensions, into multiple tensors vertically accor ing to `indices_or_sections`.
`vstack`	Stack tensors in sequence vertically (row wise).
`where`	Return a tensor of elements selecte from either `input` or `other`, epen ing on `condition`.

Accelerators

Within the PyTorch repo, we efine an “Accelerator” as a `torch.device` that is eing use alongsi e a CP to spee up computation. These evice use an asynchronous execution scheme, using `torch.Stream` an `torch.Event` as their main way to perform synchronization. We also assume that only one such accelerator can e availa le at once on a given host. This allows us to use the current accelerator as the efault evice for relevant concepts such as pinne memory, Stream evice_type, FSDP, etc.

As of to ay, accelerator evices are (in no particular or er) “C DA”, “MTIA”, “XP ”, an Private se1 (many evice not in the PyTorch repo itself).

`Stream`	An in-or er queue of executing the respective tasks asynchronously in first in first out (FIFO) or er.
`Event`	Query an recor Stream status to i entify or control epen encies across Stream an measure timing.

Generators

`Generator`	Creates an returns a generator o ect that manages the state of the algorithm which pro uces pseu o ran om num ers.

Ran om sam ling

`seed`	Sets the see for generating ran om num ers to a non-eterministic ran om num er on all evices.
`manual_seed`	Sets the see for generating ran om num ers on all evices.
`initial_seed`	Returns the initial see for generating ran om num ers as a Python *long*.
`get_rng_state`	Returns the ran om num er generator state as a *torch.ByteTensor*.
`set_rng_state`	Sets the ran om num er generator state.

`bernoulli`	Draws inary ran om num ers (0 or 1) from a Bernoulli istri ution.
`multinomial`	Returns a tensor where each row contains `num_samples` in ices sample from the multinomial (a stricter efinition woul e multivariate, refer to `torch.distributions.multinomial.Multinomial` for more etails) pro a ility istri ution locate in the correspon ing row of tensor `input`.
`normal`	Returns a tensor of ran om num ers rawn from separate normal istri utions whose mean an stan ar eviation are given.
`poisson`	Returns a tensor of the same size as `input` with each element sample from a Poisson istri ution with rate parameter given y the correspon ing element in `input` i.e.,
`rand`	Returns a tensor fille with ran om num ers from a uniform istri ution on the interval $[0, 1)$
`rand_like`	Returns a tensor with the same size as `input` that is fille with ran om num ers from a uniform istri ution on the interval $[0, 1)$.
`randint`	Returns a tensor fille with ran om integers generate uniformly etween `low` (inclusive) an `high` (exclusive).
`randint_like`	Returns a tensor with the same shape as Tensor `input` fille with ran om integers generate uniformly etween `low` (inclusive) an `high` (exclusive).
`randn`	Returns a tensor fille with ran om num ers from a normal istri ution with mean *0* an variance *1* (also calle the stan ar normal istri ution).
`randn_like`	Returns a tensor with the same size as `input` that is fille with ran om num ers from a normal istri ution with mean 0 an variance 1.
`randperm`	Returns a ran om permutation of integers from `0` to `n - 1`.

In- lace ran om sam ling

There are a few more in-place ran om sampling functions efine on Tensors as well. Click through to refer to their ocumentation:

- `torch.Tensor.bernoulli_()` - in-place version of `torch.bernoulli()`
- `torch.Tensor.cauchy_()` - num ers rawn from the Cauchy istri ution
- `torch.Tensor.exponential_()` - num ers rawn from the exponential istri ution
- `torch.Tensor.geometric_()` - elements rawn from the geometric istri ution
- `torch.Tensor.log_normal_()` - samples from the log-normal istri ution
- `torch.Tensor.normal_()` - in-place version of `torch.normal()`
- `torch.Tensor.random_()` - num ers sample from the iscrete uniform istri ution
- `torch.Tensor.uniform_()` - num ers sample from the continuous uniform istri ution

Quasi-ran om sam ling

`quasirandom.SobolEngine`	The `torch.quasirandom.SobolEngine` is an engine for generating (scram le) So ol sequences.

Serialization

`save`	Saves an o ect to a isk file.
`load`	Loa s an o ect save with `torch.save()` from a file.

Parallelism

`get_num_threads`	Returns the num er of threa s use for parallelizing CP operations
`set_num_threads`	Sets the num er of threa s use for intraop parallelism on CP .
`get_num_interop_threads`	Returns the num er of threa s use for inter-op parallelism on CP (e.g.
`set_num_interop_threads`	Sets the num er of threa s use for interop parallelism (e.g.

Locally isa ling gra ient com utation

The context managers `torch.no_grad()`, `torch.enable_grad()`, an `torch.set_grad_enabled()` are helpful for locally isa ling an ena ling gra ient computation. See Locally isa ling gra ient computation for more etails on their usage. These context managers are

```
>>> x = torch.zeros(1, requires_grad=True)
>>> with torch.no_grad():
...     y = x * 2
>>> y.requires_grad
False

>>> is_train = False
>>> with torch.set_grad_enabled(is_train):
...     y = x * 2
>>> y.requires_grad
False

>>> torch.set_grad_enabled(True)  # this can also be used as a function
>>> y = x * 2
>>> y.requires_grad
True

>>> torch.set_grad_enabled(False)
>>> y = x * 2
>>> y.requires_grad
False
```

`no_grad`	Context-manager that disables gradient calculation.
`enable_grad`	Context-manager that enables gradient calculation.
`autograd.grad_mode.set_grad_enabled`	Context-manager that sets gradient calculation on or off.
`is_grad_enabled`	Returns True if grad mode is currently enabled.
`autograd.grad_mode.inference_mode`	Context-manager that enables or disables inference mode.
`is_inference_mode_enabled`	Returns True if inference mode is currently enabled.

Math operations

Constants

`inf`	A floating-point positive infinity. Alias for `math.inf`.
`nan`	A floating-point "not a number" value. This value is not a legal number. Alias for `math.nan`.

Pointwise Ops

`abs`	Computes the absolute value of each element in `input`.
`absolute`	Alias for `torch.abs()`
`acos`	Computes the inverse cosine of each element in `input`.
`arccos`	Alias for `torch.acos()`.
`acosh`	Returns a new tensor with the inverse hyperbolic cosine of the elements of `input`.
`arccosh`	Alias for `torch.acosh()`.
`add`	Adds `other`, scaled by `alpha`, to `input`.
`addcdiv`	Performs the element-wise division of `tensor1` by `tensor2`, multiplies the result by the scalar `value` and adds it to `input`.
`addcmul`	Performs the element-wise multiplication of `tensor1` by `tensor2`, multiplies the result by the scalar `value` and adds it to `input`.
`angle`	Computes the element-wise angle (in radians) of the given `input` tensor.
`asin`	Returns a new tensor with the arcsine of the elements of `input`.
`arcsin`	Alias for `torch.asin()`.
`asinh`	Returns a new tensor with the inverse hyperbolic sine of the elements of `input`.
`arcsinh`	Alias for `torch.asinh()`.
`atan`	Returns a new tensor with the arctangent of the elements of `input`.

`atanh`	Returns a new tensor with the inverse hyper olic tangent of the elements of `input`.
`arctanh`	Alias for `torch.atanh()`.
`atan2`	Element-wise arctangent of $\text{input}_i / \text{other}_i$ with consi eration of the qua rant.
`arctan2`	Alias for `torch.atan2()`.
`bitwise_not`	Computes the itwise NOT of the given input tensor.
`bitwise_and`	Computes the itwise AND of `input` an `other`.
`bitwise_or`	Computes the itwise OR of `input` an `other`.
`bitwise_xor`	Computes the itwise XOR of `input` an `other`.
`bitwise_left_shift`	Computes the left arithmetic shift of `input` y `other` its.
`bitwise_right_shift`	Computes the right arithmetic shift of `input` y `other` its.
`ceil`	Returns a new tensor with the ceil of the elements of `input`, the smallest integer greater than or equal to each element.
`clamp`	Clamps all elements in `input` into the range [`min`, `max`].
`clip`	Alias for `torch.clamp()`.
`conj_physical`	Computes the element-wise con ugate of the given `input` tensor.
`copysign`	Create a new floating-point tensor with the magnitu e of `input` an the sign of `other`, elementwise.
`cos`	Returns a new tensor with the cosine of the elements of `input`.
`cosh`	Returns a new tensor with the hyper olic cosine of the elements of `input`.
`deg2rad`	Returns a new tensor with each of the elements of `input` converte from angles in egrees to ra ians.
`div`	Divi es each element of the input `input` y the correspon ing element of `other`.
`divide`	Alias for `torch.div()`.
`digamma`	Alias for `torch.special.digamma()`.
`erf`	Alias for `torch.special.erf()`.
`erfc`	Alias for `torch.special.erfc()`.
`erfinv`	Alias for `torch.special.erfinv()`.
`exp`	Returns a new tensor with the exponential of the elements of the input tensor `input`.
`exp2`	Alias for `torch.special.exp2()`.
`expm1`	Alias for `torch.special.expm1()`.
`fake_quantize_per_channel_affine`	Returns a new tensor with the ata in `input` fake quantize per channel using `scale`, `zero_point`, `quant_min` an `quant_max`, across the channel specifie y `axis`.
`fake_quantize_per_tensor_affine`	Returns a new tensor with the ata in `input` fake quantize using `scale`, `zero_point`, `quant_min` an `quant_max`.

`float_power`	Raises `input` to the power of `exponent`, elementwise, in ou le precision.
`floor`	Returns a new tensor with the floor of the elements of `input`, the largest integer less than or equal to each element.
`floor_divide`	
`fmod`	Applies C++'s st ::fmo entrywise.
`frac`	Computes the fractional portion of each element in `input`.
`frexp`	Decomposes `input` into mantissa an exponent tensors such that $\text{input} = \text{mantissa} \times 2^{\text{exponent}}$.
`gradient`	Estimates the gra ient of a function $g : \mathbb{R}^n \to \mathbb{R}$ in one or more imensions using the secon -or er accurate central ifferences metho an either first or secon or er estimates at the oun aries.
`imag`	Returns a new tensor containing imaginary values of the `self` tensor.
`ldexp`	Multiplies `input` y 2 ** `other`.
`lerp`	Does a linear interpolation of two tensors `start` (given y `input`) an `end` ase on a scalar or tensor `weight` an returns the resulting `out` tensor.
`lgamma`	Computes the natural logarithm of the a solute value of the gamma function on `input`.
`log`	Returns a new tensor with the natural logarithm of the elements of `input`.
`log10`	Returns a new tensor with the logarithm to the ase 10 of the elements of `input`.
`log1p`	Returns a new tensor with the natural logarithm of (1 + `input`).
`log2`	Returns a new tensor with the logarithm to the ase 2 of the elements of `input`.
`logaddexp`	Logarithm of the sum of exponentiations of the inputs.
`logaddexp2`	Logarithm of the sum of exponentiations of the inputs in ase-2.
`logical_and`	Computes the element-wise logical AND of the given input tensors.
`logical_not`	Computes the element-wise logical NOT of the given input tensor.
`logical_or`	Computes the element-wise logical OR of the given input tensors.
`logical_xor`	Computes the element-wise logical XOR of the given input tensors.
`logit`	Alias for `torch.special.logit()`.
`hypot`	Given the legs of a right triangle, return its hypotenuse.
`i0`	Alias for `torch.special.i0()`.
`igamma`	Alias for `torch.special.gammainc()`.
`igammac`	Alias for `torch.special.gammaincc()`.
`mul`	Multiplies `input` y `other`.
`multiply`	Alias for `torch.mul()`.
	Alias for `torch.special.multigammaln()`

	`input` with the values specifie y `nan`, `posinf`, an `neginf`, respectively.
`neg`	Returns a new tensor with the negative of the elements of `input`.
`negative`	Alias for `torch.neg()`
`nextafter`	Return the next floating-point value after `input` towar s `other`, elementwise.
`polygamma`	Alias for `torch.special.polygamma()`.
`positive`	Returns `input`.
`pow`	Takes the power of each element in `input` with `exponent` an returns a tensor with the result.
`quantized_batch_norm`	Applies atch normalization on a 4D (NCHW) quantize tensor.
`quantized_max_pool1d`	Applies a 1D max pooling over an input quantize tensor compose of several input planes.
`quantized_max_pool2d`	Applies a 2D max pooling over an input quantize tensor compose of several input planes.
`rad2deg`	Returns a new tensor with each of the elements of `input` converte from angles in ra ians to egrees.
`real`	Returns a new tensor containing real values of the `self` tensor.
`reciprocal`	Returns a new tensor with the reciprocal of the elements of `input`
`remainder`	Computes Python's mo ulus operation entrywise.
`round`	Roun s elements of `input` to the nearest integer.
`rsqrt`	Returns a new tensor with the reciprocal of the square-root of each of the elements of `input`.
`sigmoid`	Alias for `torch.special.expit()`.
`sign`	Returns a new tensor with the signs of the elements of `input`.
`sgn`	This function is an extension of torch.sign() to complex tensors.
`signbit`	Tests if each element of `input` has its sign it set or not.
`sin`	Returns a new tensor with the sine of the elements of `input`.
`sinc`	Alias for `torch.special.sinc()`.
`sinh`	Returns a new tensor with the hyper olic sine of the elements of `input`.
`softmax`	Alias for `torch.nn.functional.softmax()`.
`sqrt`	Returns a new tensor with the square-root of the elements of `input`.
`square`	Returns a new tensor with the square of the elements of `input`.
`sub`	Su tracts `other`, scale y `alpha`, from `input`.
`subtract`	Alias for `torch.sub()`.
`tan`	Returns a new tensor with the tangent of the elements of `input`.
`tanh`	Returns a new tensor with the hyper olic tangent of the elements of `input`.

`trunc`	Returns a new tensor with the truncate integer values of the elements of `input`.
`xlogy`	Alias for `torch.special.xlogy()`.

Re uction O s

`argmax`	Returns the in ices of the maximum value of all elements in the `input` tensor.
`argmin`	Returns the in ices of the minimum value(s) of the flattene tensor or along a imension
`amax`	Returns the maximum value of each slice of the `input` tensor in the given imension(s) `dim`.
`amin`	Returns the minimum value of each slice of the `input` tensor in the given imension(s) `dim`.
`aminmax`	Computes the minimum an maximum values of the `input` tensor.
`all`	Tests if all elements in `input` evaluate to *True*.
`any`	Tests if any element in `input` evaluates to *True*.
`max`	Returns the maximum value of all elements in the `input` tensor.
`min`	Returns the minimum value of all elements in the `input` tensor.
`dist`	Returns the p-norm of (`input` - `other`)
`logsumexp`	Returns the log of summe exponentials of each row of the `input` tensor in the given imension `dim`.
`mean`	
`nanmean`	Computes the mean of all *non-NaN* elements along the specifie imensions.
`median`	Returns the me ian of the values in `input`.
`nanmedian`	Returns the me ian of the values in `input`, ignoring `NaN` values.
`mode`	Returns a name tuple `(values, indices)` where `values` is the mo e value of each row of the `input` tensor in the given imension `dim`, i.e. a value which appears most often in that row, an `indices` is the in ex location of each mo e value foun .
`norm`	Returns the matrix norm or vector norm of a given tensor.
`nansum`	Returns the sum of all elements, treating Not a Num ers (NaNs) as zero.
`prod`	Returns the pro uct of all elements in the `input` tensor.
`quantile`	Computes the q-th quantiles of each row of the `input` tensor along the imension `dim`.
`nanquantile`	This is a variant of `torch.quantile()` that "ignores" `NaN` values, computing the quantiles `q` as if `NaN` values in `input` i not exist.
`std`	Calculates the stan ar eviation over the imensions specifie y `dim`.
`std_mean`	Calculates the stan ar eviation an mean over the imensions specifie y `dim`.
`sum`	Returns the sum of all elements in the `input` tensor.
`unique`	Returns the unique elements of the input tensor.
`unique_consecutive`	Eliminates all ut the first element from every consecutive group of equivalent elements.

`var_mean`	Calculates the variance an mean over the imensions specifie y `dim`.
`count_nonzero`	Counts the num er of non-zero values in the tensor `input` along the given `dim`.

Com arison O s

`allclose`	This function checks if `input` an `other` satisfy the con ition:
`argsort`	Returns the in ices that sort a tensor along a given imension in ascen ing or er y value.
`eq`	Computes element-wise equality
`equal`	`True` if two tensors have the same size an elements, `False` otherwise.
`ge`	Computes $\text{input} \geq \text{other}$ element-wise.
`greater_equal`	Alias for `torch.ge()`.
`gt`	Computes $\text{input} > \text{other}$ element-wise.
`greater`	Alias for `torch.gt()`.
`isclose`	Returns a new tensor with oolean elements representing if each element of `input` is ″close″ to the correspon ing element of `other`.
`isfinite`	Returns a new tensor with oolean elements representing if each element is *finite* or not.
`isin`	Tests if each element of `elements` is in `test_elements`.
`isinf`	Tests if each element of `input` is infinite (positive or negative infinity) or not.
`isposinf`	Tests if each element of `input` is positive infinity or not.
`isneginf`	Tests if each element of `input` is negative infinity or not.
`isnan`	Returns a new tensor with oolean elements representing if each element of `input` is NaN or not.
`isreal`	Returns a new tensor with oolean elements representing if each element of `input` is real-value or not.
`kthvalue`	Returns a name tuple `(values, indices)` where `values` is the `k` th smallest element of each row of the `input` tensor in the given imension `dim`.
`le`	Computes $\text{input} \leq \text{other}$ element-wise.
`less_equal`	Alias for `torch.le()`.
`lt`	Computes $\text{input} < \text{other}$ element-wise.
`less`	Alias for `torch.lt()`.
`maximum`	Computes the element-wise maximum of `input` an `other`.
`minimum`	Computes the element-wise minimum of `input` an `other`.
`fmax`	Computes the element-wise maximum of `input` an `other`.
`fmin`	Computes the element-wise minimum of `input` an `other`.
`ne`	Computes $\text{input} \neq \text{other}$ element-wise.

`sort`	Sorts the elements of the `input` tensor along a given imension in ascen ing or er y value.
`topk`	Returns the `k` largest elements of the given `input` tensor along a given imension.
`msort`	Sorts the elements of the `input` tensor along its first imension in ascen ing or er y value.

S ectral O s

`stft`	Short-time Fourier transform (STFT).
`istft`	Inverse short time Fourier Transform.
`bartlett_window`	Bartlett win ow function.
`blackman_window`	Blackman win ow function.
`hamming_window`	Hamming win ow function.
`hann_window`	Hann win ow function.
`kaiser_window`	Computes the Kaiser win ow with win ow length `window_length` an shape parameter `beta`.

Other O erations

`atleast_1d`	Returns a 1- imensional view of each input tensor with zero imensions.
`atleast_2d`	Returns a 2- imensional view of each input tensor with zero imensions.
`atleast_3d`	Returns a 3- imensional view of each input tensor with zero imensions.
`bincount`	Count the frequency of each value in an array of non-negative ints.
`block_diag`	Create a lock iagonal matrix from provi e tensors.
`broadcast_tensors`	Broa casts the given tensors accor ing to Broa casting semantics.
`broadcast_to`	Broa casts `input` to the shape `shape`.
`broadcast_shapes`	Similar to `broadcast_tensors()` ut for shapes.
`bucketize`	Returns the in ices of the uckets to which each value in the `input` elongs, where the oun aries of the uckets are set y `boundaries`.
`cartesian_prod`	Do cartesian pro uct of the given sequence of tensors.
`cdist`	Computes atche the p-norm istance etween each pair of the two collections of row vectors.
`clone`	Returns a copy of `input`.
`combinations`	Compute com inations of length r of the given tensor.
`corrcoef`	Estimates the Pearson pro uct-moment correlation coefficient matrix of the varia les given y the `input` matrix, where rows are the varia les an columns are the o servations.
`cov`	Estimates the covariance matrix of the varia les given y the `input` matrix, where rows are the varia les an columns are the o servations.
`cross`	Returns the cross pro uct of vectors in imension `dim` of `input` an `other`.
	Returns a name tuple `(values, indices)` where `values` is the

`cummin`	Returns a name tuple `(values, indices)` where `values` is the cumulative minimum of elements of `input` in the imension `dim`.
`cumprod`	Returns the cumulative pro uct of elements of `input` in the imension `dim`.
`cumsum`	Returns the cumulative sum of elements of `input` in the imension `dim`.
`diag`	• If `input` is a vector (1-D tensor), then returns a 2-D square tensor
`diag_embed`	Creates a tensor whose iagonals of certain 2D planes (specifie y `dim1` an `dim2`) are fille y `input`.
`diagflat`	• If `input` is a vector (1-D tensor), then returns a 2-D square tensor
`diagonal`	Returns a partial view of `input` with the its iagonal elements with respect to `dim1` an `dim2` appen e as a imension at the en of the shape.
`diff`	Computes the n-th forwar ifference along the given imension.
`einsum`	Sums the pro uct of the elements of the input `operands` along imensions specifie using a notation ase on the Einstein summation convention.
`flatten`	Flattens `input` y reshaping it into a one- imensional tensor.
`flip`	Reverse the or er of an n-D tensor along given axis in ims.
`fliplr`	Flip tensor in the left/right irection, returning a new tensor.
`flipud`	Flip tensor in the up/ own irection, returning a new tensor.
`kron`	Computes the Kronecker pro uct, enote y ⊗, of `input` an `other`.
`rot90`	Rotate an n-D tensor y 90 egrees in the plane specifie y ims axis.
`gcd`	Computes the element-wise greatest common ivisor (GCD) of `input` an `other`.
`histc`	Computes the histogram of a tensor.
`histogram`	Computes a histogram of the values in a tensor.
`histogramdd`	Computes a multi- imensional histogram of the values in a tensor.
`meshgrid`	Creates gri s of coor inates specifie y the 1D inputs in *attr*:tensors.
`lcm`	Computes the element-wise least common multiple (LCM) of `input` an `other`.
`logcumsumexp`	Returns the logarithm of the cumulative summation of the exponentiation of elements of `input` in the imension `dim`.
`ravel`	Return a contiguous flattene tensor.
`renorm`	Returns a tensor where each su -tensor of `input` along imension `dim` is normalize such that the *p*-norm of the su -tensor is lower than the value `maxnorm`
`repeat_interleave`	Repeat elements of a tensor.
`roll`	Roll the tensor `input` along the given imension(s).
`searchsorted`	Fin the in ices from the *innermost* imension of `sorted_sequence` such that, if the correspon ing values in `values` were inserte efore the in ices, when sorte , the or er of the correspon ing *innermost* imension within `sorted_sequence` woul e preserve .

`trace`	Returns the sum of the elements of the iagonal of the input 2-D matrix.
`tril`	Returns the lower triangular part of the matrix (2-D tensor) or atch of matrices `input`, the other elements of the result tensor `out` are set to 0.
`tril_indices`	Returns the in ices of the lower triangular part of a `row`- y- `col` matrix in a 2- y-N Tensor, where the first row contains row coor inates of all in ices an the secon row contains column coor inates.
`triu`	Returns the upper triangular part of a matrix (2-D tensor) or atch of matrices `input`, the other elements of the result tensor `out` are set to 0.
`triu_indices`	Returns the in ices of the upper triangular part of a `row` y `col` matrix in a 2- y-N Tensor, where the first row contains row coor inates of all in ices an the secon row contains column coor inates.
`unflatten`	Expan s a imension of the input tensor over multiple imensions.
`vander`	Generates a Van ermon e matrix.
`view_as_real`	Returns a view of `input` as a real tensor.
`view_as_complex`	Returns a view of `input` as a complex tensor.
`resolve_conj`	Returns a new tensor with materialize con ugation if `input`'s con ugate it is set to *True*, else returns `input`.
`resolve_neg`	Returns a new tensor with materialize negation if `input`'s negative it is set to *True*, else returns `input`.

BLAS an LAPACK O erations

`addbmm`	Performs a atch matrix-matrix pro uct of matrices store in `batch1` an `batch2`, with a re uce a step (all matrix multiplications get accumulate along the first imension).
`addmm`	Performs a matrix multiplication of the matrices `mat1` an `mat2`.
`addmv`	Performs a matrix-vector pro uct of the matrix `mat` an the vector `vec`.
`addr`	Performs the outer-pro uct of vectors `vec1` an `vec2` an a s it to the matrix `input`.
`baddbmm`	Performs a atch matrix-matrix pro uct of matrices in `batch1` an `batch2`.
`bmm`	Performs a atch matrix-matrix pro uct of matrices store in `input` an `mat2`.
`chain_matmul`	Returns the matrix pro uct of the N 2-D tensors.
`cholesky`	Computes the Cholesky ecomposition of a symmetric positive- efinite matrix A or for atches of symmetric positive- efinite matrices.
`cholesky_inverse`	Computes the inverse of a complex Hermitian or real symmetric positive- efinite matrix given its Cholesky ecomposition.
`cholesky_solve`	Computes the solution of a system of linear equations with complex Hermitian or real symmetric positive- efinite lhs given its Cholesky ecomposition.
`dot`	Computes the ot pro uct of two 1D tensors.
`geqrf`	This is a low-level function for calling LAPACK's geqrf irectly.
`ger`	Alias of `torch.outer()`.
`inner`	Computes the ot pro uct for 1D tensors.
`inverse`	Alias for `torch.linalg.inv()`

`logdet`	Calculates log eterminant of a square matrix or atches of square matrices.
`slogdet`	Alias for `torch.linalg.slogdet()`
`lu`	Computes the L factorization of a matrix or atches of matrices `A`.
`lu_solve`	Returns the L solve of the linear system $Ax = b$ using the partially pivote L factorization of A from `lu_factor()`.
`lu_unpack`	npacks the L ecomposition returne y `lu_factor()` into the *P, L, U* matrices.
`matmul`	Matrix pro uct of two tensors.
`matrix_power`	Alias for `torch.linalg.matrix_power()`
`matrix_exp`	Alias for `torch.linalg.matrix_exp()`.
`mm`	Performs a matrix multiplication of the matrices `input` an `mat2`.
`mv`	Performs a matrix-vector pro uct of the matrix `input` an the vector `vec`.
`orgqr`	Alias for `torch.linalg.householder_product()`.
`ormqr`	Computes the matrix-matrix multiplication of a pro uct of Househol er matrices with a general matrix.
`outer`	Outer pro uct of `input` an `vec2`.
`pinverse`	Alias for `torch.linalg.pinv()`
`qr`	Computes the QR ecomposition of a matrix or a atch of matrices `input`, an returns a name tuple (Q, R) of tensors such that $\text{input} = QR$ with Q eing an orthogonal matrix or atch of orthogonal matrices an R eing an upper triangular matrix or atch of upper triangular matrices.
`svd`	Computes the singular value ecomposition of either a matrix or atch of matrices `input`.
`svd_lowrank`	Return the singular value ecomposition `(U, S, V)` of a matrix, atches of matrices, or a sparse matrix A such that $A \approx U \operatorname{diag}(S) V^{\mathrm{H}}$.
`pca_lowrank`	Performs linear Principal Component Analysis (PCA) on a low-rank matrix, atches of such matrices, or sparse matrix.
`lobpcg`	Fin the k largest (or smallest) eigenvalues an the correspon ing eigenvectors of a symmetric positive efinite generalize eigenvalue pro lem using matrix-free LOBPCG metho s.
`trapz`	Alias for `torch.trapezoid()`.
`trapezoid`	Computes the trapezoi al rule along `dim`.
`cumulative_trapezoid`	Cumulatively computes the trapezoi al rule along `dim`.
`triangular_solve`	Solves a system of equations with a square upper or lower triangular inverti le matrix A an multiple right-han si es b.
`vdot`	Computes the ot pro uct of two 1D vectors along a imension.

Foreach O erations

WARNING

This API is in eta an su ect to future changes. Forwar -mo e AD is not supporte .

`_foreach_abs`	Apply `torch.abs()` to each Tensor of the input list.
`_foreach_abs_`	Apply `torch.abs()` to each Tensor of the input list.

`_foreach_acos_`	Apply `torch.acos()` to each Tensor of the input list.
`_foreach_asin`	Apply `torch.asin()` to each Tensor of the input list.
`_foreach_asin_`	Apply `torch.asin()` to each Tensor of the input list.
`_foreach_atan`	Apply `torch.atan()` to each Tensor of the input list.
`_foreach_atan_`	Apply `torch.atan()` to each Tensor of the input list.
`_foreach_ceil`	Apply `torch.ceil()` to each Tensor of the input list.
`_foreach_ceil_`	Apply `torch.ceil()` to each Tensor of the input list.
`_foreach_cos`	Apply `torch.cos()` to each Tensor of the input list.
`_foreach_cos_`	Apply `torch.cos()` to each Tensor of the input list.
`_foreach_cosh`	Apply `torch.cosh()` to each Tensor of the input list.
`_foreach_cosh_`	Apply `torch.cosh()` to each Tensor of the input list.
`_foreach_erf`	Apply `torch.erf()` to each Tensor of the input list.
`_foreach_erf_`	Apply `torch.erf()` to each Tensor of the input list.
`_foreach_erfc`	Apply `torch.erfc()` to each Tensor of the input list.
`_foreach_erfc_`	Apply `torch.erfc()` to each Tensor of the input list.
`_foreach_exp`	Apply `torch.exp()` to each Tensor of the input list.
`_foreach_exp_`	Apply `torch.exp()` to each Tensor of the input list.
`_foreach_expm1`	Apply `torch.expm1()` to each Tensor of the input list.
`_foreach_expm1_`	Apply `torch.expm1()` to each Tensor of the input list.
`_foreach_floor`	Apply `torch.floor()` to each Tensor of the input list.
`_foreach_floor_`	Apply `torch.floor()` to each Tensor of the input list.
`_foreach_log`	Apply `torch.log()` to each Tensor of the input list.
`_foreach_log_`	Apply `torch.log()` to each Tensor of the input list.
`_foreach_log10`	Apply `torch.log10()` to each Tensor of the input list.
`_foreach_log10_`	Apply `torch.log10()` to each Tensor of the input list.
`_foreach_log1p`	Apply `torch.log1p()` to each Tensor of the input list.
`_foreach_log1p_`	Apply `torch.log1p()` to each Tensor of the input list.
`_foreach_log2`	Apply `torch.log2()` to each Tensor of the input list.
`_foreach_log2_`	Apply `torch.log2()` to each Tensor of the input list.
`_foreach_neg`	Apply `torch.neg()` to each Tensor of the input list.

`_foreach_tan`	Apply `torch.tan()` to each Tensor of the input list.
`_foreach_tan_`	Apply `torch.tan()` to each Tensor of the input list.
`_foreach_sin`	Apply `torch.sin()` to each Tensor of the input list.
`_foreach_sin_`	Apply `torch.sin()` to each Tensor of the input list.
`_foreach_sinh`	Apply `torch.sinh()` to each Tensor of the input list.
`_foreach_sinh_`	Apply `torch.sinh()` to each Tensor of the input list.
`_foreach_round`	Apply `torch.round()` to each Tensor of the input list.
`_foreach_round_`	Apply `torch.round()` to each Tensor of the input list.
`_foreach_sqrt`	Apply `torch.sqrt()` to each Tensor of the input list.
`_foreach_sqrt_`	Apply `torch.sqrt()` to each Tensor of the input list.
`_foreach_lgamma`	Apply `torch.lgamma()` to each Tensor of the input list.
`_foreach_lgamma_`	Apply `torch.lgamma()` to each Tensor of the input list.
`_foreach_frac`	Apply `torch.frac()` to each Tensor of the input list.
`_foreach_frac_`	Apply `torch.frac()` to each Tensor of the input list.
`_foreach_reciprocal`	Apply `torch.reciprocal()` to each Tensor of the input list.
`_foreach_reciprocal_`	Apply `torch.reciprocal()` to each Tensor of the input list.
`_foreach_sigmoid`	Apply `torch.sigmoid()` to each Tensor of the input list.
`_foreach_sigmoid_`	Apply `torch.sigmoid()` to each Tensor of the input list.
`_foreach_trunc`	Apply `torch.trunc()` to each Tensor of the input list.
`_foreach_trunc_`	Apply `torch.trunc()` to each Tensor of the input list.
`_foreach_zero_`	Apply `torch.zero()` to each Tensor of the input list.

tilities

`compiled_with_cxx11_abi`	Returns whether PyTorch was uilt with _GLIBCXX_ SE_CXX11_ABI=1
`result_type`	Returns the `torch.dtype` that woul result from performing an arithmetic operation on the provi e input tensors.
`can_cast`	Determines if a type conversion is allowe un er PyTorch casting rules escri e in the type promotion ocumentation.
`promote_types`	Returns the `torch.dtype` with the smallest size an scalar kin that is not smaller nor of lower kin than either *type1* or *type2*.
`use_deterministic_algorithms`	Sets whether PyTorch operations must use " eterministic" algorithms.
`are_deterministic_algorithms_enabled`	Returns True if the glo al eterministic flag is turne on.
`is_deterministic_algorithms_warn_only_enabled`	Returns True if the glo al eterministic flag is set to warn only.
`set_deterministic_debug_mode`	Sets the e ug mo e for eterministic operations.

`set_float32_matmul_precision`	Sets the internal precision of float32 matrix multiplications.
`get_float32_matmul_precision`	Returns the current value of float32 matrix multiplication precision.
`set_warn_always`	When this flag is False (efault) then some PyTorch warnings may only appear once per process.
`get_device_module`	Returns the mo ule associate with a given evice(e.g., torch. evice('cu a'), "mtia:0", "xpu", ...).
`is_warn_always_enabled`	Returns True if the glo al warn_always flag is turne on.
`vmap`	vmap is the vectorizing map; `vmap(func)` returns a new function that maps `func` over some imension of the inputs.
`_assert`	A wrapper aroun Python's assert which is sym olically tracea le.

Sym olic Num ers

CLASS torch.SymInt(*node*) [SO RCE]

Like an int (inclu ing magic metho s), ut re irects all operations on the wrappe no e. This is use in particular to sym olically recor operations in the sym olic shape workflow.

as_integer_ratio() [SO RCE]

Represent this int as an exact integer ratio

Return ty e

Tuple[*SymInt*, int]

CLASS torch.SymFloat(*node*) [SO RCE]

Like an float (inclu ing magic metho s), ut re irects all operations on the wrappe no e. This is use in particular to sym olically recor operations in the sym olic shape workflow.

as_integer_ratio() [SO RCE]

Represent this float as an exact integer ratio

Return ty e

Tuple[int, int]

con ugate() [SO RCE]

Returns the complex con ugate of the float.

Return ty e

SymFloat

hex() [SO RCE]

Returns the hexa ecimal representation of the float.

Return ty e

str

is_integer() [SO RCE]

Return True if the float is an integer.

CLASS torch.SymBool(*node*) [SO RCE]

Like an ool (inclu ing magic metho s), ut re irects all operations on the wrappe no e. This is use in particular to sym olically recor operations in the sym olic shape workflow.

nlike regular ools, regular oolean operators will force extra guar s instea of sym olically evaluate. se the itwise operators instea to han le this.

`sym_float`	SymInt-aware utility for float casting.
`sym_fresh_size`	
`sym_int`	SymInt-aware utility for int casting.
`sym_max`	SymInt-aware utility for max which avoi s ranching on a < .
`sym_min`	SymInt-aware utility for min().
`sym_not`	SymInt-aware utility for logical negation.
`sym_ite`	
`sym_sum`	N-ary a which is faster to compute for long lists than iterate inary a ition.

• WARNING

This feature is a prototype an may have compati ility reaking changes in the future.

export generate /export /in ex

Control Flow

• WARNING

This feature is a prototype an may have compati ility reaking changes in the future.

`cond`	Con itionally applies *true_fn* or *false_fn*.

O timizations

`compile`	Optimizes given mo el/function using TorchDynamo an specifie acken .

torch.compile ocumentation

O erator Tags

CLASS torch.Tag

Mem ers:

core

ata_ epen ent_output

ynamic_output_shape

flexi le_layout

generate

inplace_view

may e_aliasing_or_mutating

nee s_fixe _stri e_or er

non eterministic_ itwise

non eterministic_see e

pointwise

pt2_compliant_tag

view_copy

PROPERTY name

‹ Previous

Next ›