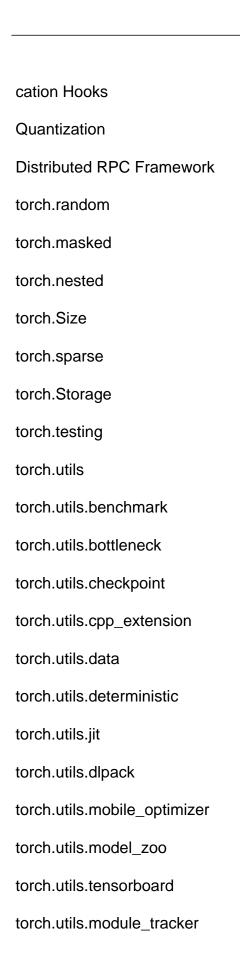
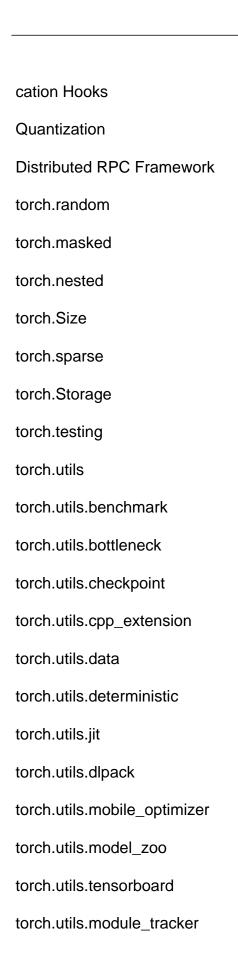
https://pytorch.org/docs/stable/torch.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch
Shortcuts
torch
The torch package contains data structures for multi-dimensional tensors and defines mathematical
operations over these tensors. Additionally, it provides many utilities for efficient serialization of
Tensors and arbitrary types, and other useful utilities.
It has a CUDA counterpart, that enables you to run your tensor computations on an NVIDIA GPU
with compute capability >= 3.0.
Tensor

https://pytorch.org/docs/stable/nn.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.nn
Shortcuts
torch.nn
These are the basic building blocks for graphs:
torch.nn
Containers
Convolution Layers
Pooling layers
Padding Layers
Non-linear Activations (weighted sum, nonlinearity)

Normalization Layers	
Recurrent Layers	
Transformer Layers	
Linear Layers	
Dropout Layers	
Sparse Layers	
Distance Functions	
Loss Functions	
Vision Layers	
Shuffle Layers	
Data	

Non-linear Activations (other)

https://pytorch.org/docs/stable/nn.functional.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.nn.functional
Shortcuts
torch.nn.functional
Convolution functions
conv1d
Applies a 1D convolution over an input signal composed of several input planes.
conv2d

Applies a 2D convolution over an input image composed of several input planes.
conv3d
Applies a 3D convolution over an input image composed of several input planes.
conv_transpose1d
Applies a 1D transposed convolution o
Applies a 1D transposed convolution o

https://pytorch.org/docs/stable/tensors.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.Tensor
Shortcuts
torch.Tensor
A torch.Tensor is a multi-dimensional matrix containing elements of a single data type.
Data types
Torch defines tensor types with the following data types:
Data type
dtype

32-bit floating point	
torch.float32 or torch.float	
64-bit floating point	
torch.float64 or torch.double	
16-bit floating point 1	
torch.float16 or torch.half	

16-bit floatin

https://pytorch.org/docs/stable/tensor_attributes.html

cation Hooks
Quantization
Distributed RPC Framework
torch.random
torch.masked
torch.nested
torch.Size
torch.sparse
torch.Storage
torch.testing
torch.utils
torch.utils.benchmark
torch.utils.bottleneck
torch.utils.checkpoint
torch.utils.cpp_extension
torch.utils.data
torch.utils.deterministic
torch.utils.jit
torch.utils.dlpack
torch.utils.mobile_optimizer
torch.utils.model_zoo
torch.utils.tensorboard
torch.utils.module_tracker

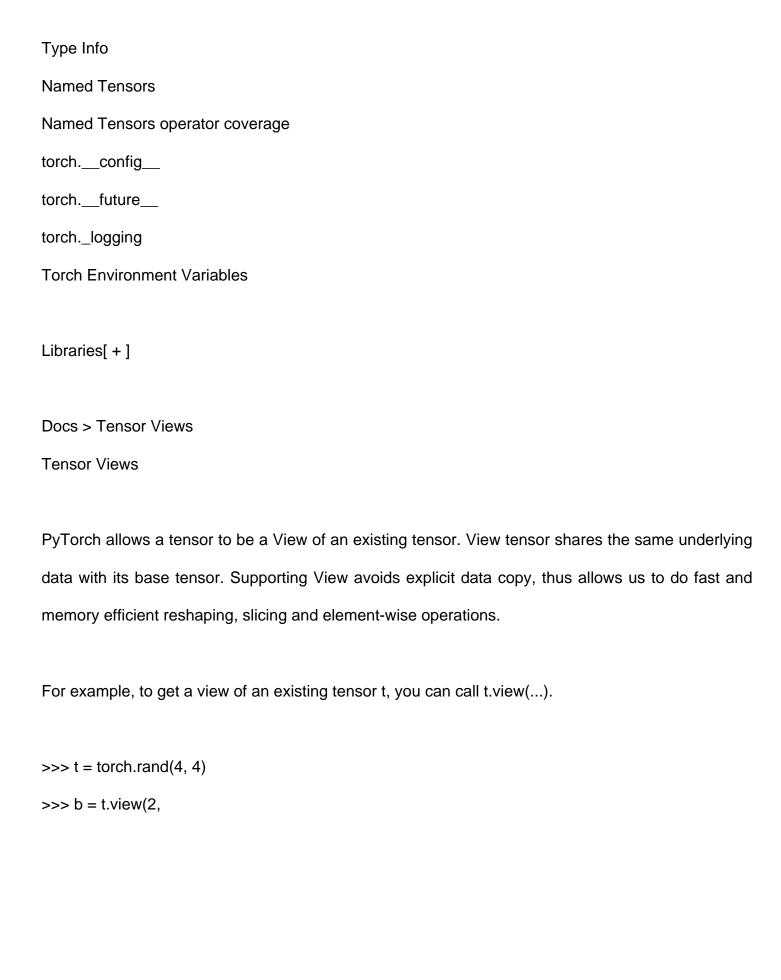
Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Tensor Attributes
Shortcuts
Tensor Attributes
Each torch.Tensor has a torch.dtype, torch.device, and torch.layout.
torch.dtype
CLASS
torch.dtype
A torch.dtype is an object that represents the data type of a torch.Tensor. PyTorch has twelve
different data types:
Data type

Legacy Constructors		
32-bit floating point		
torch.float32 or torch.float		
torch.*.FloatTensor		
64-bit floating p		

dtype

https://pytorch.org/docs/stable/tensor_view.html





https://pytorch.org/docs/stable/amp.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Automatic Mixed Precision package - torch.amp
Shortcuts
Automatic Mixed Precision package - torch.amp
torch.amp provides convenience methods for mixed precision, where some operations use the

torch.float32 (float) datatype and other operations use lower precision floating point datatype

(lower_precision_fp): torch.float16 (half) or torch.bfloat16. Some ops, like linear layers and

convolutions, are much faster

https://pytorch.org/docs/stable/autograd.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Automatic differentiation package - torch.autograd
Shortcuts
Automatic differentiation package - torch.autograd
torch.autograd provides classes and functions implementing automatic differentiation of arbitrary
scalar valued functions.
It requires minimal changes to the existing code - you only need to declare Tensor s for which
gradients should be computed with the requires_grad=True keyword. As of now, we on

https://pytorch.org/docs/stable/library.html

Landing Page for more details on how to effectively use these APIs.

Testing custom ops

Use torch.library.opcheck() to test custom ops for incorrect usage of the Python torch.library and/or C++ TORCH_LIBRARY APIs. Also, if your operator supports training, use torch.autograd.gradcheck() to test that the gradients are mathematically correct.

torch.library.opcheck(op, args, kwargs=None, *, test_utils=('test_schema', 'test_autograd_registration', 'test_faketensor', 'test_aot_dispatch_dynamic'), raise_exception=True)
[SOURCE]

Given an operator and some sample arguments, tests if the operator is registered correctly.

That is, when you use the torch.library/TORCH_LIBRARY APIs to create a custom op, you specified metadata (e.g. mutability info) about the custom op and these APIs require that the functions you pass them satisfy certain properties (e.g. no data pointer access in the fake/meta/abstract kernel) opcheck tests these metadata and properties.

Concretely, we test the following:

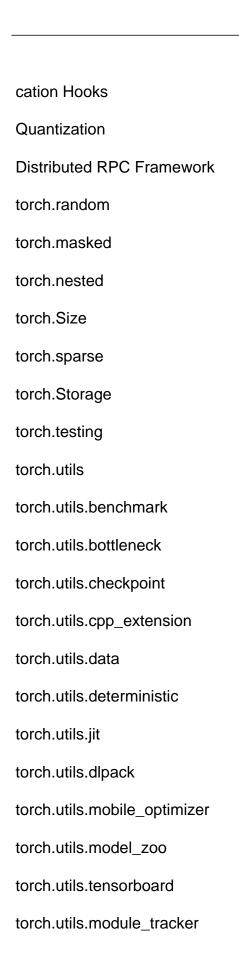
https://pytorch.org/docs/stable/cpu.html



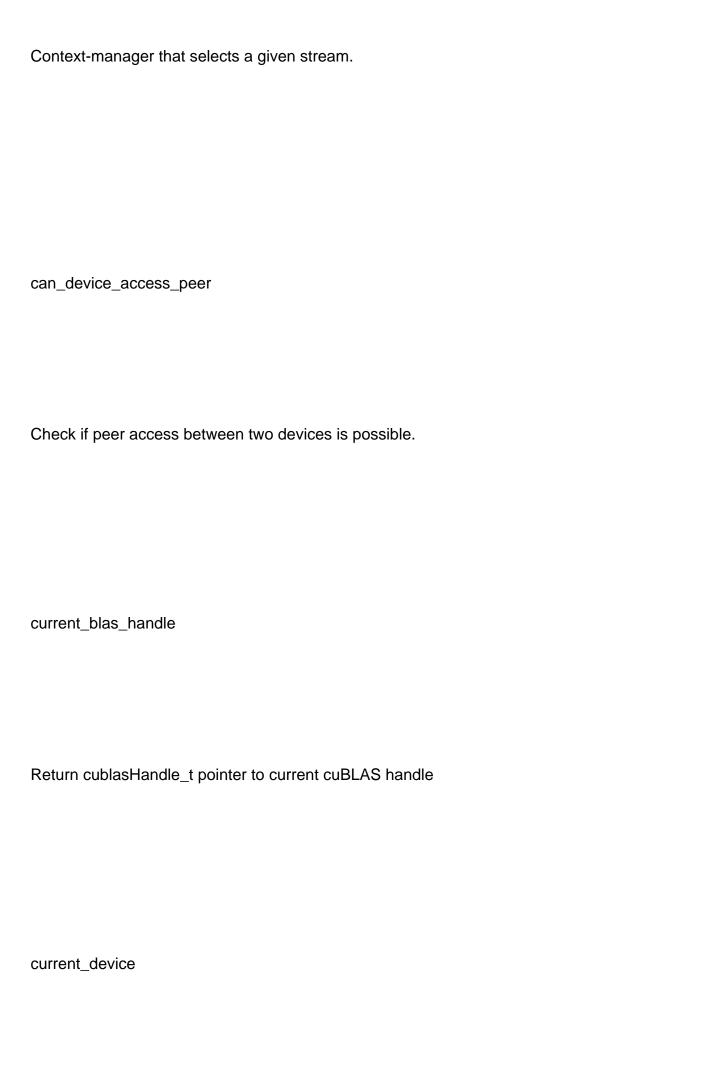
Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.cpu
Shortcuts
torch.cpu
This package implements abstractions found in torch.cuda to facilitate writing device-agnostic code.
current_device
Returns current device for cpu.
current_stream

Returns the currently selected Stream for a given device.
is_available
Returns a bool indicating if CPU is currently available.
. totallie a 2001 maioamig ii Ci C lo darromiy available.
synchronize
Waits for all kernels in all streams on the CPU dev
waits for all kernels in all streams on the CPO dev

https://pytorch.org/docs/stable/cuda.html

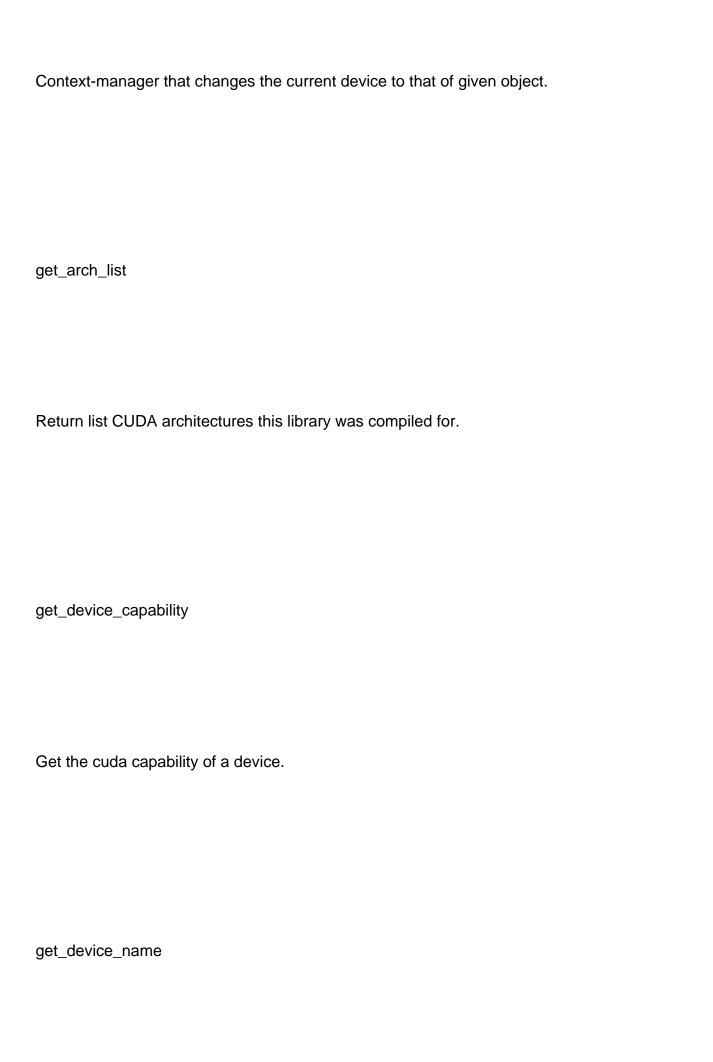


Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.cuda
Shortcuts
torch.cuda
This package adds support for CUDA tensor types.
It implements the same function as CPU tensors, but they utilize GPUs for computation.
It is lazily initialized, so you can always import it, and use is_available() to determine if your system
supports CUDA.
CUDA semantics has more details about working with CUDA.
StreamContext



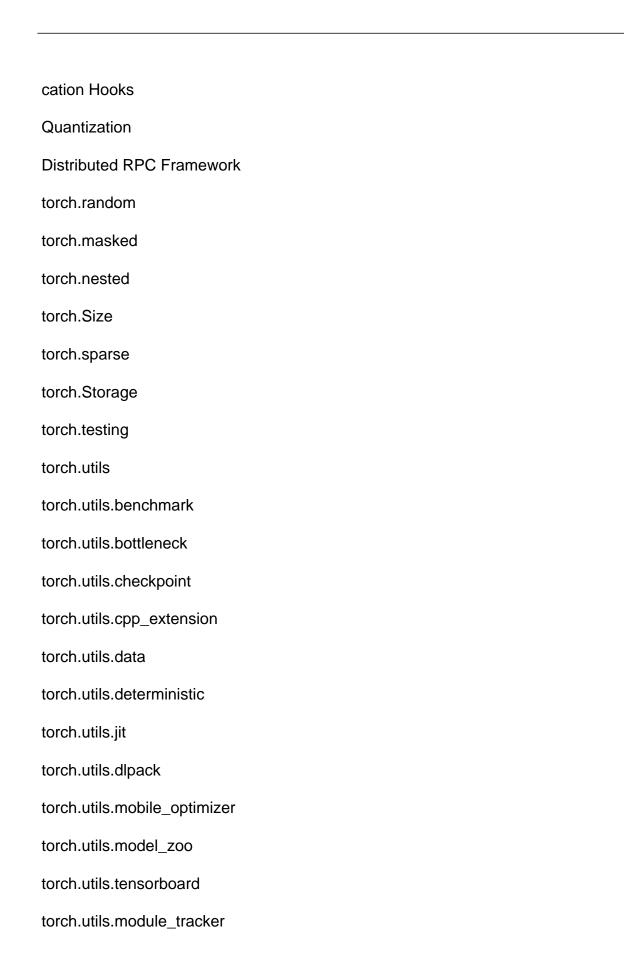
Return the index of a currently selected device.
current_stream
Return the currently selected Stream for a given device.
cudart
Retrieves the CUDA runtime API module.
default_stream

Return the default Stream for a given device.
device
Context-manager that changes the selected device.
device_count
Return the number of GPUs available.
device_of



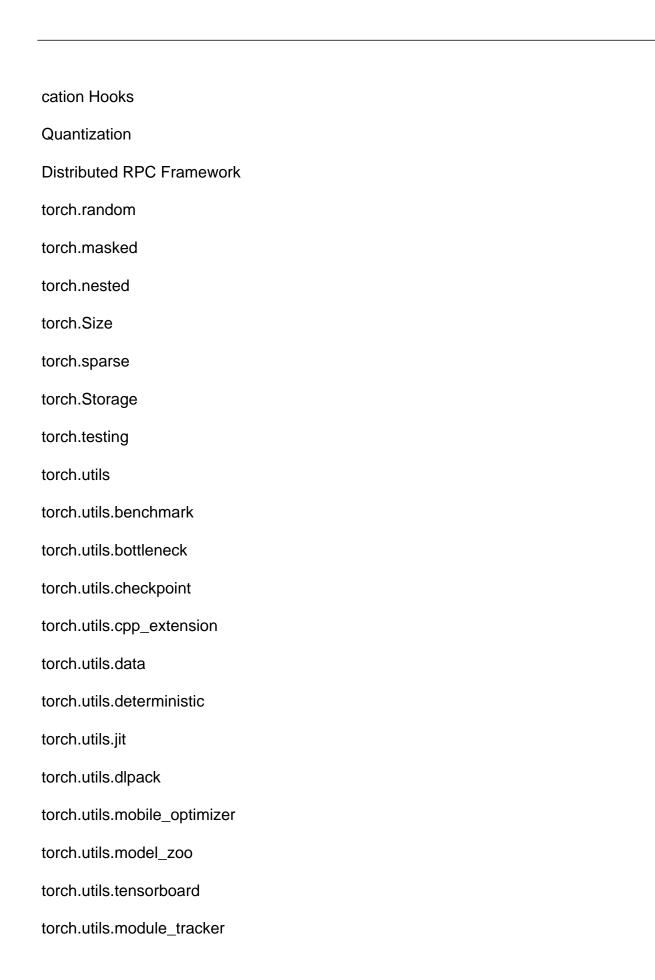
Get the name of a device.
get_device_properties
Get the properties of a device.
get_gencode_flags
Return NVCC gencode flags this library was compiled wi

https://pytorch.org/docs/stable/torch_cuda_memory.html



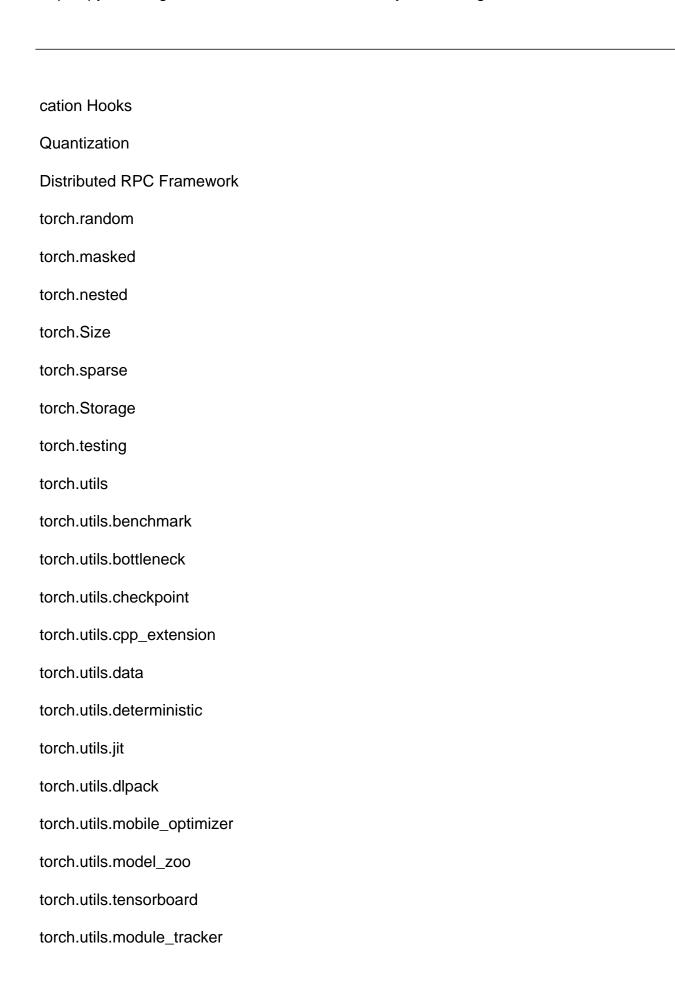
Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Understanding CUDA Memory Usage
Shortcuts
Understanding CUDA Memory Usage
To debug CUDA memory use, PyTorch provides a way to generate memory snapshots that record
the state of allocated CUDA memory at any point in time, and optionally record the history of
allocation events that led up to that snapshot.
The generated snapshots can then be drag and dropped onto the interactiver viewer hosted at
pytorch.org/me

https://pytorch.org/docs/stable/torch_cuda_memory.html#generating-a-snapshot



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Understanding CUDA Memory Usage
Shortcuts
Understanding CUDA Memory Usage
To debug CUDA memory use, PyTorch provides a way to generate memory snapshots that record
the state of allocated CUDA memory at any point in time, and optionally record the history of
allocation events that led up to that snapshot.
The generated snapshots can then be drag and dropped onto the interactiver viewer hosted at
pytorch.org/me

https://pytorch.org/docs/stable/torch_cuda_memory.html#using-the-visualizer



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Understanding CUDA Memory Usage
Shortcuts
Understanding CUDA Memory Usage
To debug CUDA memory use, PyTorch provides a way to generate memory snapshots that record
the state of allocated CUDA memory at any point in time, and optionally record the history of
allocation events that led up to that snapshot.
The generated snapshots can then be drag and dropped onto the interactiver viewer hosted at
pytorch.org/me

https://pytorch.org/docs/stable/torch_cuda_memory.html#snapshot-api-reference



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Understanding CUDA Memory Usage
Shortcuts
Understanding CUDA Memory Usage
To debug CUDA memory use, PyTorch provides a way to generate memory snapshots that record
the state of allocated CUDA memory at any point in time, and optionally record the history of
allocation events that led up to that snapshot.
The generated snapshots can then be drag and dropped onto the interactiver viewer hosted at
pytorch.org/me

https://pytorch.org/docs/stable/export.html



Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > torch.export
Shortcuts
torch.export
WARNING
This feature is a prototype under active development and there WILL BE BREAKING CHANGES in
the future.
Overview
torch.export.export() takes an arbitrary Python callable (a torch.nn.Module, a function or a method)
and produces a traced graph representing only the Tensor computation of the function in an
Ahead-of-Time (AOT) fashion, which can subsequently be execute

Type Info

https://pytorch.org/docs/stable/distributed.html



Type Info
Named Tensors
Named Tensors operator coverage
torchconfig
torchfuture
torchlogging
Torch Environment Variables
Libraries[+]
Docs > Distributed communication package - torch.distributed
Shortcuts
Distributed communication package - torch.distributed
NOTE
Please refer to PyTorch Distributed Overview for a brief introduction to all features related to
distributed training.
Backends
torch.distributed supports three built-in backends, each with different capabilities. The table below
shows which functions are available for use with CPU / CU