



# Denglin Hamming™ V2

## PydINNE API

DL-DG/SW-054A-01

2024-04-25

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

## 商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

## 通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

## Change History

Version	Change description
01	Initial version

CONTENTS

<b>1</b>	<b>Library API</b>	<b>3</b>
1.1	Class Hierarchy . . . . .	3
1.2	File Hierarchy . . . . .	4
1.3	Full API . . . . .	4

**LIBRARY API**

## 1.1 Class Hierarchy

- *Namespace dlnne*
  - *Class Builder*
  - *Class BuilderFlag*
  - *Class ClusterConfig*
  - *Class DataType*
  - *Class Engine*
  - *Class ExecutionContext*
  - *Class Format*
  - *Class GpuTarget*
  - *Class ModelFormat*
  - *Class Network*
  - *Class NetworkDefinitionCreationFlag*
  - *Class OptimizationProfile*
  - *Class OptProfileSelector*
  - *Class OptProfileType*
  - *Class Parser*
  - *Class TensorIOMode*
  - *Class WeightShareMode*

## 1.2 File Hierarchy

- file\_dlnne.py

## 1.3 Full API

### 1.3.1 Namespaces

#### Namespace dlnne

**Contents**

- *Classes*
- *Functions*

#### Classes

- *Class Builder*
- *Class BuilderFlag*
- *Class ClusterConfig*
- *Class DataType*
- *Class Engine*
- *Class ExecutionContext*
- *Class Format*
- *Class GpuTarget*
- *Class ModelFormat*
- *Class Network*
- *Class NetworkDefinitionCreationFlag*
- *Class OptimizationProfile*
- *Class OptProfileSelector*
- *Class OptProfileType*
- *Class Parser*
- *Class TensorIOMode*
- *Class WeightShareMode*

## Functions

- *Function `dlnne::deserialize`*

### 1.3.2 Classes and Structs

#### Class Builder

- Defined in file `dlnne.py`

#### Class Documentation

**class** `dlnne.Builder`

Build an engine from network definition.

#### Public Functions

`__enter__` (*self*)

`__exit__` (*self*, *exc\_type*, *exc\_val*, *exc\_tb*)

`__del__` (*self*)

`__init__` (*self*)

`max_batch_size` (*self*)

get the maximum batch size

**Return** The maximum batch size

`max_batch_size` (*self*, *batch\_size*)

set the maximum batch size

**Return** None

#### Parameters

- *batch\_size*: The maximum batch size can be used at execution time

`create_network` (*self*, *flags=0*)

Create a network definition object.

**Return** A new network object

#### Parameters

- *flags*: *NetworkDefinitionCreationFlag* combined using bitwise OR.

`build_engine` (*self*, *network*)

Build *Engine*.

**Return** An engine from network definition

#### Parameters

- *network*: *Network* object

**set\_flag** (*self*, *flag*)

Set a single build mode flag.

**Return** None

**Parameters**

- *flag*: The flag added to the already enabled flags

**get\_flag** (*self*, *flag*)

Return true if the build mode flag is set.

**Return** Return true if flag is set, false if unset

**Parameters**

- *flag*: The flag to be queried

**set\_flags** (*self*, *flags*)

Set the build mode flags to turn on builder options for this network.

This function will override the previous flags

**Return** None

**Parameters**

- *flags*: The flags are listed in the *BuilderFlag* enum

**get\_flags** (*self*)

Get the build mode flags for this builder config.

The default value is 0

**Return** The build options as a bitmask

**create\_optimization\_profile** (*self*)

Create a new optimization profile.

**Return** A new *OptimizationProfile* object

**build\_serialize\_network** (*self*, *network*)

Build and serialize a network for the given *Network*.

This function allows building and serialization of a network without creating an engine.

**Return** Bytes contains a serialized engine

**Parameters**

- *network*: *Network* definition



## Public Members

`handle`  
`max_batch_size`  
`config`

## Class BuilderFlag

- Defined in file\_dlnne.py

## Class Documentation

**class** `dlnne.BuilderFlag`  
*Builder* options.

### Public Static Attributes

`spm_alloc = pydlnne.BuilderFlag.kSpmAlloc`  
 [DEPRECATED]  
`user_const = pydlnne.BuilderFlag.kUserConst`  
 Use user-allocated const memory instead.  
`slz_ignore_weights = pydlnne.BuilderFlag.kSlzIgnoreWeights`  
 Ignore weights in *Engine.serialize()*

## Class ClusterConfig

- Defined in file\_dlnne.py

## Class Documentation

**class** `dlnne.ClusterConfig`

Weight share mode corresponds to cluster configuration. When the weight share mode is set to single, cluster0, cluster1, cluster2 and cluster3 are valid cluster configuration. When the weight share mode is set to share2, cluster01, cluster23, cluster02, cluster13, cluster03, cluster12 are valid. When the weight share mode is set to share4, cluster0123 is the only choice.

### Public Static Attributes

`cluster0 = pydlnne.ClusterConfig.kCluster0`  
`cluster1 = pydlnne.ClusterConfig.kCluster1`  
`cluster2 = pydlnne.ClusterConfig.kCluster2`  
`cluster3 = pydlnne.ClusterConfig.kCluster3`  
`cluster01 = pydlnne.ClusterConfig.kCluster01`  
`cluster23 = pydlnne.ClusterConfig.kCluster23`

```

cluster02 = pydlnne.ClusterConfig.kCluster02
cluster13 = pydlnne.ClusterConfig.kCluster13
cluster03 = pydlnne.ClusterConfig.kCluster03
cluster12 = pydlnne.ClusterConfig.kCluster12
cluster0123 = pydlnne.ClusterConfig.kCluster0123
cluster0 = pydlnne.ClusterConfig.kCluster0
[DEPRECATED].
Please use cluster0 instead

cluster1 = pydlnne.ClusterConfig.kCluster1
[DEPRECATED].
Please use cluster1 instead

cluster2 = pydlnne.ClusterConfig.kCluster2
[DEPRECATED].
Please use cluster2 instead

cluster3 = pydlnne.ClusterConfig.kCluster3
[DEPRECATED].
Please use cluster3 instead

cluster01 = pydlnne.ClusterConfig.kCluster01
[DEPRECATED].
Please use cluster01 instead

cluster23 = pydlnne.ClusterConfig.kCluster23
[DEPRECATED].
Please use cluster23 instead

cluster0123 = pydlnne.ClusterConfig.kCluster0123
[DEPRECATED].
Please use cluster0123 instead

```

## Class DataType

- Defined in file\_dlnne.py

## Class Documentation

```

class dlnne.DataType
    DataType.

```

### Public Static Attributes

```

HALF = pydlnne.DataType.kFLOAT16
FLOAT = pydlnne.DataType.kFLOAT32
FLOAT64 = pydlnne.DataType.kFLOAT64
INT8 = pydlnne.DataType.kINT8
INT16 = pydlnne.DataType.kINT16
INT32 = pydlnne.DataType.kINT32
INT64 = pydlnne.DataType.kINT64
UINT8 = pydlnne.DataType.kUINT8
UINT16 = pydlnne.DataType.kUINT16
UINT32 = pydlnne.DataType.kUINT32
UINT64 = pydlnne.DataType.kUINT64
BOOL = pydlnne.DataType.kBOOL

```

### Class Engine

- Defined in file\_dlnne.py

### Class Documentation

**class** dlnne.Engine

An engine for executing inference on a built network, with functionally unsafe features.

#### Public Functions

**\_\_enter\_\_** (*self*)

**\_\_init\_\_** (*self*)

**\_\_exit\_\_** (*self*, *exc\_type*, *exc\_val*, *exc\_tb*)

**\_\_del\_\_** (*self*)

**num\_bindings** (*self*)

Get the number of binding indices.

**Return** The total bindings over all profiles

**num\_io\_tensors** (*self*)

Experimental.

Get the number of binding indices for single profile

**Return** The bindings for single profile

**max\_batch\_size** (*self*)

Get the maximum batch size which can be used for inference.

**Return** The maximum batch size which can be used for inference.

**num\_optimization\_profiles** (*self*)

Get the number of optimization profiles define for this engine.

**Return** The number of optimization profiles define for this engine

**has\_implicit\_batch\_dimension** (*self*)

Query whether the engine was built with an implicit batch dimension.

**Return** True if has implicit batch

**binding\_is\_input** (*self*, \**arg*)

Overloaded function 1.binding\_is\_input(*self*, index: int):

Determine whether a binding is an input binding

index: Binding index

returns: True if binding is input

2.binding\_is\_input(*self*, name: str):

Determine whether a binding is an input binding

name: Binding name

returns: True if binding is input

**get\_binding\_index** (*self*, *name*)

Retrieve the binding index for a named tensor.

**Return** The binding index for the named tensor, or -1 if the name is not found.

**Parameters**

- name: The tensor name

**get\_binding\_shape** (*self*, *index*)

Get the dimensions of a binding.

**Return** The dimensions of the binding if the index is in range, otherwise (1,). Has -1 for any dimension with a dynamic value

**Parameters**

- index: The binding index

**get\_binding\_name** (*self*, *index*)

Retrieve the name corresponding to a binding index.

**Return** The name corresponding to the index, or none if the index is out of range

**Parameters**

- index: The binding index

**create\_execution\_context** (*self*, *config*=pydlnne.ClusterConfig.kCluster0)

Create an execution context.

**Return** A new *ExecutionContext* object

**Parameters**

- `config`: The config in *ClusterConfig*

**get\_binding\_dtype** (*self*, \**arg*)

Overloaded function 1.get\_binding\_dtype(*self*, *index*: int):

Determine the required data type for a buffer from its binding index

*index*: Binding index

returns: The type of the data in the buffer

2.get\_binding\_dtype(*self*, *name*: str):

Determine the required data type for a buffer from its binding index

*name*: Binding name

returns: The type of the data in the buffer

**serialize** (*self*)

Serialize the network to a stream.

**Return** Bytes contains the serialized engine

**shrink** (*self*)

Release the resources allocated by tensor compiler.

Thus, UploadConst and CreateExecutionContext cannot be called anymore after calling this function.

**Return** None

**set\_const\_memory** (*self*, *mems*)

[DEPRECATED].

Please use *set\_const\_memory\_ext()* instead

**upload\_const** (*self*, *mems*)

[DEPRECATED].

Please use *upload\_const\_ext()* instead

**set\_const\_memory\_ext** (*self*, *hash\_key*, *mem*)

Set per-const const memory.

**Return** False if *hash\_key* cannot be found in this engine, otherwise true

**Parameters**

- *hash\_key*: Hash key of the const
- *mem*: constant memory

**upload\_const\_ext** (*self*, *hash\_key*, *mem*)

Upload const mem to device.

**Return** False if *hash\_key* cannot be found in this engine, otherwise true

**Parameters**

- *hash\_key*: Hash key of the const
- *mem*: constant memory

**get\_const\_info\_ext** (*self*, *hash\_key*, *mem\_size*)

Query all const buffer info.

**Return** True if get success

**Parameters**

- *hash\_key*: A hash\_key list to load per-const mem hashKey
- *mem\_size*: A mem\_size list to load per-const mem size

**serialize\_without\_const\_ext** (*self*, *hash\_key\_list*, *size*)

Serialize engine without const in hash\_key\_list.

**Return** Bytes for serialize data

**Parameters**

- *hash\_key\_list*: hash key list of the const not serialize
- *size*: size of hash\_key\_list

**get\_per\_cluster\_const\_memory\_size** (*self*)

[DEPRECATED].

Please use [get\\_const\\_info\\_ext\(\)](#) instead

**get\_per\_cluster\_device\_memory\_size** (*self*)

[DEPRECATED].

Please use [get\\_device\\_memory\\_size\(\)](#) instead

**get\_device\_memory\_size** (*self*)

Query the device memory size required by the execution context.

**Return** Return total device memory size is sufficient for the max batch size

**get\_profile\_shape** (*self*, \**arg*)

Overloaded function 1.get\_profile\_shape(*self*, *index*: int):

Get the minimum/optimum/maximum dimensions list for a particular binding under an oprimization profile

*index*: Binding index

returns: A List[Dims] of length 3, containing the minimum, optimum, and maximum shapes, in that order

2.get\_profile\_shape(*self*, *name*: str):

Get the minimum/optimum/maximum dimensions list for a particular binding under an oprimization profile

*name*: Binding name

returns: A List[Dims] of length 3, containing the minimum, optimum, and maximum shapes, in that order

**get\_tensor\_dtype** (*self*, *name*)

Experimental.

Determine the required data type for a buffer from its tensor name

**Return** The type of the data in the buffer

**Parameters**

- *name*: The name of an input or output tensor

**get\_tensor\_name** (*self, index*)

Experimental.

Return the name of an IO tensor

**Return** Tensor name

**Parameters**

- *index*: The tensor index

**get\_tensor\_profile\_shape** (*self, name, index*)

Experimental.

Get the minimum / optimum / maximum dimensions list for an input tensor given its name under an optimization profile

**Return** A List[Dims] of length 3, containing the minimum, optimum, and maximum shapes, in that order

**Parameters**

- *name*: The name of an input or output tensor
- *index*: The profile index

**get\_tensor\_shape** (*self, name*)

Experimental.

Get extent of an input or output tensor

**Return** Extent of the tensor

**Parameters**

- *name*: The name of an input or output tensor

**get\_tensor\_mode** (*self, name*)

Experimental.

Determine whether a tensor is an input or output tensor

**Return** Return `kIOModeInput` if `tensor_name` is an input, `kIOModeOutput` if `tensor_name` is an output, or `kIOModeNone` if neither

**Parameters**

- *name*: The name of an input or output tensor

## Class ExecutionContext

- Defined in `file_dlnne.py`

## Class Documentation

**class** `dlnne.ExecutionContext`

Context for executing inference using an engine.

## Public Functions

**\_\_enter\_\_** (*self*)

**\_\_init\_\_** (*self, engine, config*)

**\_\_del\_\_** (*self*)

**\_\_exit\_\_** (*self, exc\_type, exc\_val, exc\_tb*)

**active\_optimization\_profile** (*self*)

Get the index of the currently selected optimization profile.

**Return** The index of the currently selected optimization profile

**execute** (*self, batch\_size, bindings*)

Synchronously execute inference on a batch.

**Return** True if execution succeeded

### Parameters

- *batch\_size*: The batch size
- *bindings*: Input and output buffers

**execute\_async** (*self, batch\_size, bindings, stream\_handle, inputConsumed=None*)

Asynchronously execute inference on a batch.

**Return** True if enqueue succeeded

### Parameters

- *batch\_size*: The batch size
- *bindings*: Input and output buffers
- *stream\_handle*: A cuda stream on which the inference kernels will be enqueued
- *inputConsumed*: An optional event which will be signaled when the input buffers can be refilled with new data

**execute\_v2** (*self, bindings*)

Synchronously execute inference on a batch.

**Return** True if execution success

### Parameters

- *bindings*: List[int(memory handle)], the list for input and output buffers

**execute\_async\_v2** (*self, bindings, stream\_handle, inputConsumed=None*)

Asynchronously execute inference.

**Return** True if enqueue success

### Parameters

- *bindings*: List[int(memory handle)], the list for input and output buffers
- *stream\_handle*: int(Stream.handle), a cuda stream on which the inference kernels will be enqueued



- `inputConsumed`: List[int(event handle)], an optional event which will be signaled when the input buffers can be refilled with new data

**set\_bindings\_shape** (*self, binding, shape*)

Set the dynamic dimensions of a binding.

**Return** True if set success

**Parameters**

- `binding`: The index of the input buffers
- `shape`: The dimension which will set for the corresponding index

**get\_bindings\_shape** (*self, index*)

Get the dynamic dimensions of a binding.

**Return** Currently selected binding dimensions

**Parameters**

- `index`: The specified binding\_index of the buffers

**set\_optimization\_profile** (*self, index*)

Select an optimization profile for the current context.

**Return** True if set success

**Parameters**

- `index`: Index of the profile

**set\_input\_shape** (*self, name, shape*)

Experimental.

Set shape of given input

**Return** True if set success

**Parameters**

- `name`: The name of an input tensor
- `shape`: The shape of an input tensor

**get\_tensor\_shape** (*self, name*)

Experimental.

Return the shape of the given input or output

**Return** Tensor shape

**Parameters**

- `name`: The name of an input or output tensor

**set\_tensor\_address** (*self, name, memory*)

Experimental.

Set memory address for given input or output tensor

**Return** True if set success

**Parameters**

- **name**: The name of an input or output tensor
- **memory**: int(memory handle) or None, the data owned by the user

**get\_tensor\_address** (*self*, *name*)

Experimental.

Get memory address bound to given input or output tensor

**Return** Memory handle(int)

**Parameters**

- **name**: The name of an input or output tensor

**infer\_shapes** (*self*)

Experimental.

Run shape calculations

**Return** Error list for tensor names(List[tensor names])

**input\_consumed\_event** (*self*)

Experimental.

Get the event associated with consuming the input.

**Return** Event handle(int)

**input\_consumed\_event** (*self*, *event*)

Experimental.

Set or reset an event which will be triggered when all input tensor buffers can be refilled

**Return** True if set success

**Parameters**

- **event**: Event object or int(event handle)

**execute\_async\_v3** (*self*, *stream*)

Experimental.

Enqueue inference on a stream

**Return** True if enqueue success

**Parameters**

- **stream**: int(Stream.handle)

## Public Members

**handle**

## Class Format

- Defined in file\_dlnne.py

## Class Documentation

**class** dlnne.Format  
Tensor formats in memory.

### Public Static Attributes

LINEAR = pydlnne.Format.kLINEAR  
NCHW = pydlnne.Format.kNCHW  
NHWC = pydlnne.Format.kNHWC

## Class GpuTarget

- Defined in file\_dlnne.py

## Class Documentation

**class** dlnne.GpuTarget  
[DEPRECATED].  
Please use *Network.set\_config()* with device-profile instead

### Public Static Attributes

CurrentGpuTarget = pydlnne.GpuTarget.kCurrentGpu  
GoldwasserL256 = pydlnne.GpuTarget.kGoldwasserL256  
GoldwasserL128 = pydlnne.GpuTarget.kGoldwasserL128  
GoldwasserUL64 = pydlnne.GpuTarget.kGoldwasserUL64  
GoldwasserUL32 = pydlnne.GpuTarget.kGoldwasserUL32

## Class ModelFormat

- Defined in file\_dlnne.py

## Class Documentation

### **class** `dlnne.ModelFormat`

Model type for *Parser.ParseBuffers()* If model is relay\_model, only same version sdk can be used in.

#### Public Static Attributes

```
onnx_model = pydlnne.ModelFormat.kOnnxModel
tensorflow_model = pydlnne.ModelFormat.kTensorflowModel
caffe_model = pydlnne.ModelFormat.kCaffeModel
relay_model = pydlnne.ModelFormat.kRelayModel
```

### Class Network

- Defined in file\_dlnne.py

## Class Documentation

### **class** `dlnne.Network`

A network definition for input to the builder.

#### Public Functions

**\_\_enter\_\_** (*self*)

**\_\_init\_\_** (*self*)

**num\_inputs** (*self*)

Get the number of inputs in the network.

**Return** The number of inputs in the network

**num\_outputs** (*self*)

Get the number of outputs in the network.

**Return** The number of outputs in the network

**name** (*self*)

Return the name associated with the network.

**Return** Return the name associated with the network

**name** (*self*, *name*)

Set the name of the network.

**Return** None

#### Parameters

- *name*: The name of the network

**has\_implicit\_batch\_dimension** (*self*)

Query whether the network was created with an implicit batch dimension.

**Return** True if implicit

**num\_optimization\_profiles** (*self*)

Get number of optimization profiles.

**Return** The number of optimization profiles

**add\_optimization\_profile** (*self, profile*)

Add an optimization profile.

**Return** The index of the optimization profile(starting from 0) if the input is valid, or -1 if the input is not valid

**Parameters**

- *profile*: The new *OptimizationProfile*

**set\_config** (*self, config*)

Set config.

**Return** True if set success

**Parameters**

- *config*: The config set for network

**\_\_exit\_\_** (*self, exc\_type, exc\_val, exc\_tb*)

**\_\_del\_\_** (*self*)

## Class NetworkDefinitionCreationFlag

- Defined in file\_dlnne.py

## Class Documentation

**class** dlnne.**NetworkDefinitionCreationFlag**

*Network* create options.

## Public Static Attributes

**kEXPLICIT\_BATCH** = pydlnne.NetworkDefinitionCreationFlag.kEXPLICIT\_BATCH

## Class OptimizationProfile

- Defined in file\_dlnne.py

## Class Documentation

**class** dlnne.OptimizationProfile

Profile for dynamic input dimensions.

### Public Functions

**\_\_bool\_\_** (*self*)

Check whether the optimization profile can be passed to an BuilderConfig object.

**Return** True if valid

**get\_shape** (*self, name*)

Get the minimum / optimum / maximum dimensions for a dynamic input tensor.

**Return** List of shape

**Parameters**

- name: The name of input tensor

**set\_shape** (*self, name, type, min, opt, max*)

Set the minimum / optimum / maximum dimensions for a dynamic input tensor with name input\_name.

**Return** True if set success

**Parameters**

- name: The name of input tensor
- type: The input type
- min: The min shape
- opt: The opt shape
- max: The max shape

**add\_json** (*self, json*)

Add an json file for data-depend.

**Return** True if add success

**Parameters**

- json: The new json file

## Class OptProfileSelector

- Defined in file\_dlnne.py

## Class Documentation

### **class** dlnne.OptProfileSelector

When setting or getting optimization profile parameters of dynamic dimensions, select whether the minimum, optimum or maximum values for these parameters are to set/get.

The minimum and maximum specify the permitted range that is supported at runtime.

### Public Static Attributes

`kOptProfileMin = pydlnne.OptProfileSelector.kOptProfileMin`

`kOptProfileOpt = pydlnne.OptProfileSelector.kOptProfileOpt`

`kOptProfileMax = pydlnne.OptProfileSelector.kOptProfileMax`

## Class OptProfileType

- Defined in file\_dlnne.py

## Class Documentation

### **class** dlnne.OptProfileType

When setting or getting optimization profile parameters of dynamic dimensions, select tensor type.

### Public Static Attributes

`kInput = pydlnne.OptProfileType.kInput`

`kDataDependOps = pydlnne.OptProfileType.kDataDependOps`

## Class Parser

- Defined in file\_dlnne.py

## Class Documentation

### **class** dlnne.Parser

Class used for parsing models described using the tensorflow graph.

## Public Functions

**\_\_enter\_\_** (*self*)

**\_\_exit\_\_** (*self, exc\_type, exc\_val, exc\_tb*)

**\_\_del\_\_** (*self*)

**\_\_init\_\_** (*self*)

**register\_input** (*self, input\_op\_name, shape, format=Format.NHWC*)

Register an input name of a tensorflow network with associated dimensions.

**Return** True if register success

### Parameters

- *input\_op\_name*: An input name of a tensorflow network
- *shape*: Input dimensions
- *format*: Input format. Optional

**register\_output** (*self, output\_op\_name*)

Register an output name of a tensorflow network.

**Return** True if register success

### Parameters

- *output\_op\_name*: An output name of a tensorflow network

**register\_user\_op** (*self, library\_name, module, first\_op\_name*)

Register the plugin library name used in a tensorflow network.

**Return** True if register success

### Parameters

- *library\_name*: The path and name of a tensorflow customer defined library
- *module*: The name and path of the python file
- *first\_op\_name*: The name of the first op defined by user. Optional

**parse** (*self, mod, network*)

Parse a tensorflow graph file.

**Return** True if parse success

### Parameters

- *mod*: The name of the tensorflow graph
- *network*: [Network](#) in which the parser will fill

**parse\_v2** (*self, mod, sub\_graph\_config, network*)

Parse a tensorflow graph file.

**Return** True if parse success

### Parameters



- `mod`: The name of the tensorflow graph
- `sub_graph_config`: Config for sub graph
- `network`: *Network* in which the parser will fill

**parse\_relay\_module** (*self, mod, network, inputs, outputs*)  
[DEPRECATED].

Not support since v2

**ParseBuffers** (*self, format, buffers, sizes, num, network*)  
Parse from memory buffers.

**Return** True if parse success

#### Parameters

- `format`: The type of the model
- `buffers`: The buffers for parse
- `sizes`: The size of buffer
- `num`: The number of buffers
- `network`: *Network* in which the parser will fill

#### Public Members

**handle**

#### Class TensorIOMode

- Defined in file\_dlnne.py

#### Class Documentation

**class** `dlnne.TensorIOMode`  
Experimental.  
Input or output mode of a tensor

#### Public Static Attributes

**NONE** = `pydlnne.TensorIOMode.kIOModeNone`  
Tensor is not an input or output.

**INPUT** = `pydlnne.TensorIOMode.kIOModeInput`  
Tensor is input to the engine.

**OUTPUT** = `pydlnne.TensorIOMode.kIOModeOutput`  
Tensor is output to the engine.

## Class WeightShareMode

- Defined in file\_dlnne.py

## Class Documentation

### **class** dlnne.WeightShareMode

Share the weights on different clusters. kSingle means executing a network on single cluster without weight share. kShare2 means sharing network weights on two clusters. kShare4 means sharing network weights on all four clusters.

#### Public Static Attributes

```
single = pydlnne.WeightShareMode.kSingle
share2 = pydlnne.WeightShareMode.kWeightShare2
share4 = pydlnne.WeightShareMode.kWeightShare4
```

## 1.3.3 Functions

### Function dlnne::deserialize

- Defined in file\_dlnne.py

## Function Documentation

`dlnne.deserialize(raw_data)`

Create engine from raw\_data.

**Return** An new *Engine* object

#### Parameters

- raw\_data: Data for engine