Extend OneDNN Operators/APIs for AI Chips

Beijing Institute of Open Source Chip (BOSC)

2025/03/06

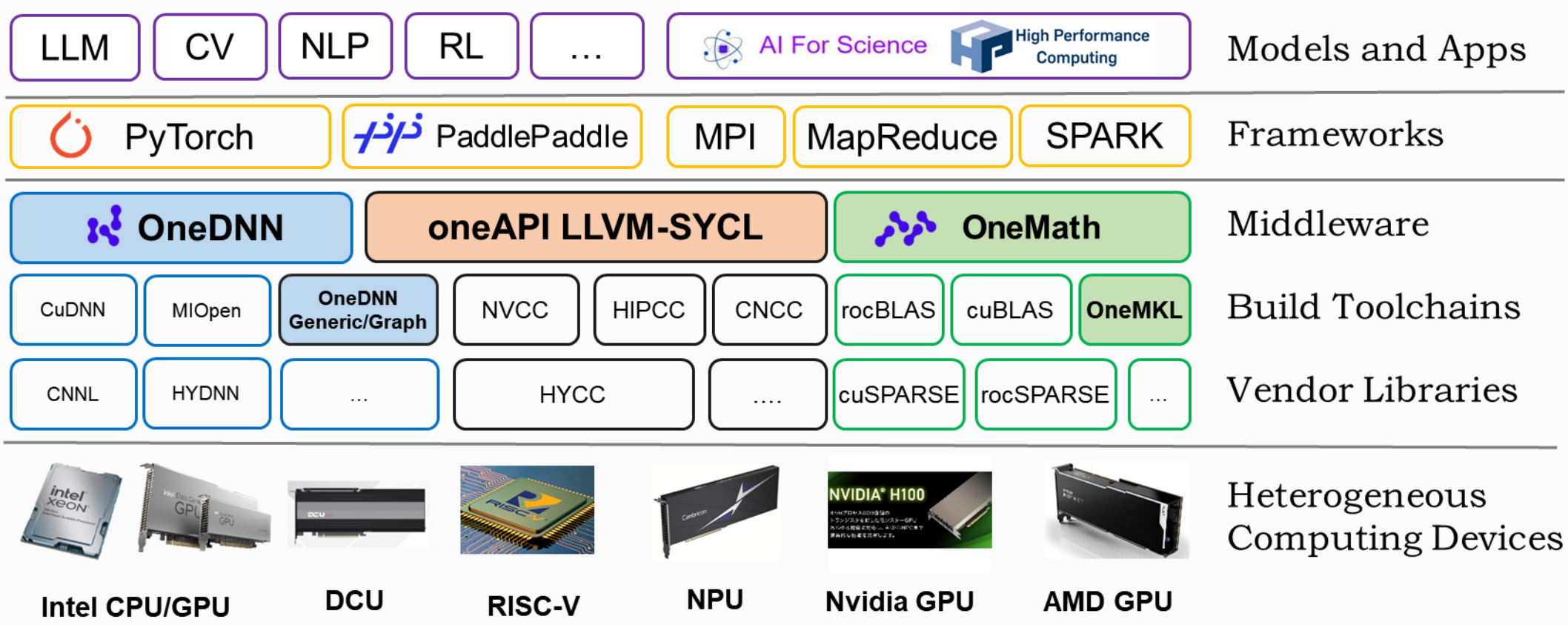
Contents:

- Background and Motivation
- Proposed Work
- Example API Implementation

Background and Motivation

Background and Motivation

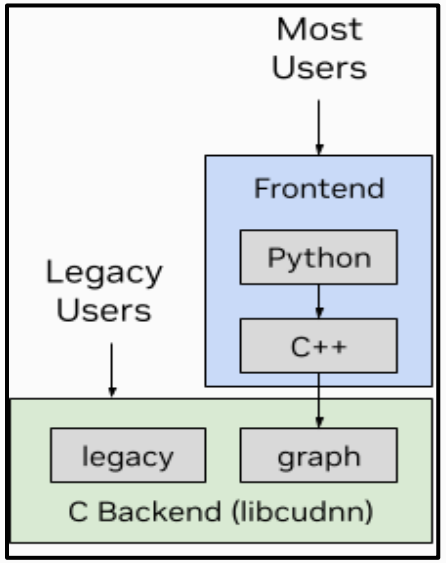
- **Building a Unified Computing Software Platform**: Leveraging heterogeneous computing power to create a platform that serves tasks of LLM, DL, AI4Science, HPC and so on;
- **Integrating OneDNN as a Backend Library**: Utilizing OneDNN as the backend library for machine learning frameworks (e.g., PyTorch, PaddlePaddle) to interface with heterogeneous computing resources;



Background and Motivation

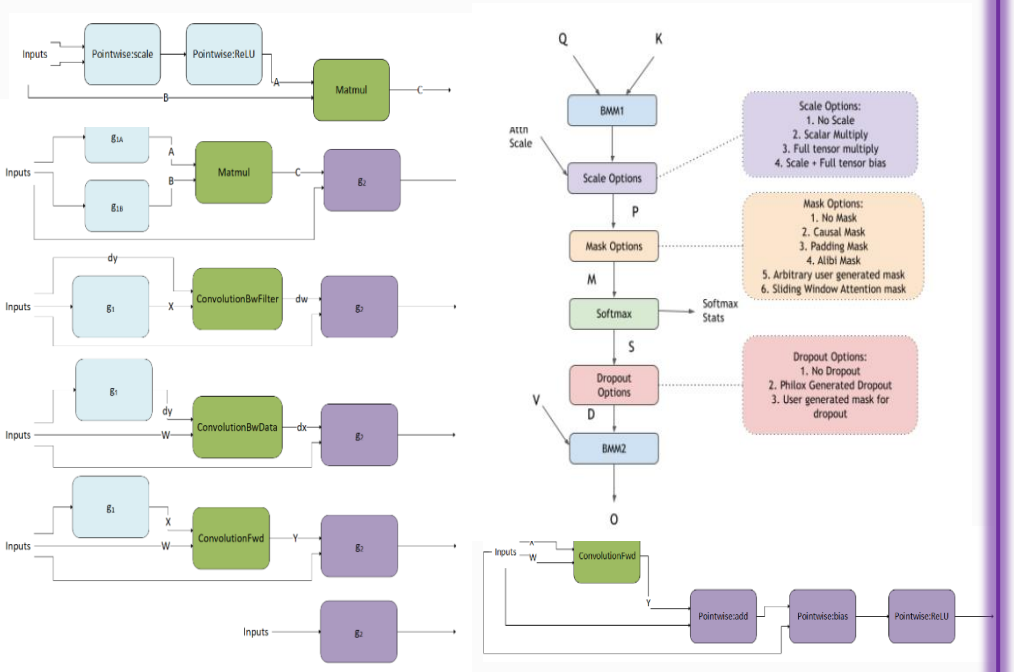
➢ Compare OneDNN Graph APIs to CuDNN Graph APIs

- CuDNN **Graph API Structure**



- CuDNN **Backend Graph API**
  - **Convolution** forward and backward, including cross-correlation
  - **Matrix multiplication**
  - **Pooling** forward and backward
  - **Softmax** forward and backward
  - **Arithmetic, mathematical, relational, and logical** pointwise operations (including various flavors of forward and backward neuron activations)
  - **Tensor transformation** functions
  - **Normalization**, like LRN, LCN, batch normalization, instance normalization, and layer normalization forward and backward

- CuDNN **Frontend API :**
  - **Operations:** *Attention, Block Scaling, Convolutions, Matmul, Normalizations, Pointwise and Reduction, Resampling, Slice*
  - **Supported Graph Patterns :**



Background and Motivation

- **CUDA API Migration Support**: According to the latest *Intel® DPC++ Compatibility Tool Developer Guide and Reference* [1], there are still **135** CuDNN APIs that have not yet received migration support.

Function	Migration Support
cudnnCopyAlgorithmDescriptor	NO
cudnnCreateAlgorithmDescriptor	NO
cudnnCreateAlgorithmPerformance	NO
cudnnCreateSpatialTransformerDescriptor	NO
cudnnCreateTensorTransformDescriptor	NO
cudnnDestroyAlgorithmDescriptor	NO
cudnnDestroyAlgorithmPerformance	NO
cudnnDestroySpatialTransformerDescriptor	NO
cudnnDestroyTensorTransformDescriptor	NO
cudnnDivisiveNormalizationForward	NO
cudnnGetAlgorithmDescriptor	NO
cudnnGetAlgorithmPerformance	NO
cudnnGetAlgorithmSpaceSize	NO
cudnnGetCallback	NO
cudnnGetProperty	NO
cudnnGetReductionIndicesSize	NO
cudnnGetTensorTransformDescriptor	NO
cudnnInitTransformDest	NO
cudnnOpsInferVersionCheck	NO
cudnnQueryRuntimeError	NO
cudnnRestoreAlgorithm	NO
cudnnSaveAlgorithm	NO
cudnnSetAlgorithmDescriptor	NO
cudnnSetAlgorithmPerformance	NO
cudnnSetCallback	NO
cudnnSetSpatialTransformerNdDescriptor	NO
cudnnSetTensorTransformDescriptor	NO

[1] https://www.intel.com/content/www/us/en/docs/dpcpp-compatibility-tool/developer-guide-reference/2025-0/api-mapping-status.html#CUDNN-API

Background and Motivation

- Compare OneDNN Graph APIs to CuDNN Graph APIs in Functionalities
  - OneDNN Graph API Structure vs CuDNN Graph API structure
  - OneDNN Graph Backend Ops vs CuDNN Backend/Frontend Graph API Ops
  - OneDNN Graph API Support Graph Patterns vs CuDNN Frontend API Ops Support Graph Patterns
  - OneDNN Primitive APIs vs CuDNN Backend Leagacy APIs
- Check if OneDNN Graph APIs works on Nvidia/AMD GPU and domestic DCU, NPU

```
(torch_onednn) lihongyang@gxn61:~/project/onednn_ex-develop/build/examples$ ./graph-sycl-single-op-partition-cpp
Graph:
 [matmul_src0] [matmul_src1]
       \          /
         matmul
            |
        [matmul_dst]
Note:
 '[]' represents a logical tensor, which refers to inputs/outputs of the graph.
Example passed on GPU.
```

- Single Op(Matmul) graph partition works on DCU

```
(base) zhangzhenling@master141-k100:~/work/onednn_ex/build/examples$ ./graph-sycl-getting-started-cpp gpu
oneDNN error caught:
        Status: unimplemented
        Message: could not create a primitive descriptor for a convolution forward propagation primitive
Example failed on GPU.
```

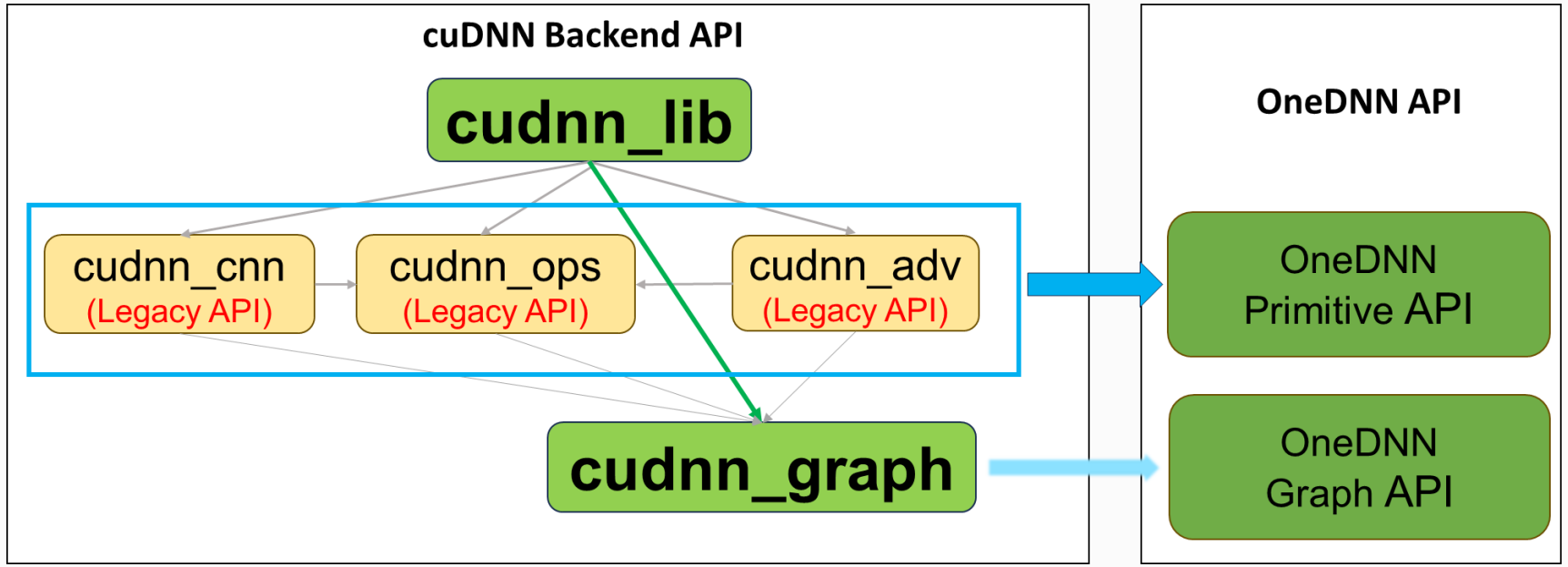
- OneDnn Graph API Convolution does NOT work on DCU

Proposed Work

Proposed Work

➢ Implement Legacy Operators/APIs

- **DivisiveNormalization**: Implement the forward and backward computations for the spatial Divisive Normalization layer.
- **MultiHeadAttn**: Implement the forward and backward computations for the multi-head attention mechanism, including derivatives with respect to inputs (Q, K, V) and trainable parameters (projection weights and biases).
- **Im2Col**: Constructs the matrix necessary to perform a forward pass of GEMM convolution.
- **CTCLoss**: Implement the Connectionist Temporal Classification (CTC) loss function and its gradient computation.



Example API Implementation

Example API Implementation

Divisive Normalization APIs: Formula

- Implement the **forward** and **backward** computations for the spatial Divisive Normalization layer.
- Supported tensor formats are NCHW for 4D and NCDHW for 5D;

Forward:

$$y_{n,c,h,w} = \left\{ k + \frac{\alpha}{n} \sum_{i=-(r-1)/2}^{(r+1)/2-1} \sum_{j=-(r-1)/2}^{(r+1)/2-1} \left(x_{n,c,h+i,w+j} - \mathrm{mean}_{n,c,h,w} \right)^2 \right\}^{-\beta} \cdot x_{n,c,h,w}$$

Backward:

$$dx_{n,c,h,w} = \omega_{n,c,h,w} - \frac{2\alpha\beta}{n} \sum_{i=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \sum_{j=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \omega_{n,c,h+i,w+j}^{-\beta-1} \left(x_{n,c,h,w} - \mathrm{mean}_{n,c,h+i,w+j} \right) x_{n,c,h+i,w+j} dy_{n,c,h,w}$$

$$d\mathrm{mean}_{n,c,h,w} = \frac{2\alpha\beta}{n} x_{n,c,h,w} \omega_{n,c,h,w}^{-\beta-1} \sum_{i=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \sum_{j=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \left(x_{n,c,h+i,w+j} - \mathrm{mean}_{n,c,h,w} \right) dy_{n,c,h,w}$$

$$\omega_{n,c,h,w} = k + \frac{\alpha}{n} \sum_{i=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \sum_{j=\frac{-r-1}{2}}^{\frac{r+1}{2}-1} \left(x_{n,c,h+i,w+j} - \mathrm{mean}_{n,c,h,w} \right)^2$$

Example API Implementation

Divisive Normalization APIs : Parameters

Forward:

$$y_{n,c,h,w} = \left\{ k + \frac{\alpha}{n} \sum_{i=-(r-1)/2}^{(r+1)/2-1} \sum_{j=-(r-1)/2}^{(r+1)/2-1} \left(x_{n,c,h+i,w+j} - \text{mean}_{n,c,h,w} \right)^2 \right\}^{-\beta} \cdot x_{n,c,h,w}$$

Parameters:

x, y: input and out tensor
k, α: scaling factors;
n: the size of normalization window, r^2 for 4D tensor and r^3 for 5D tensor
r: normalization window width
β: value of the beta power parameter in the normalization formula. By default, this value is set to 0.75

Example API Implementation

Divisive Normalization APIs : Description

APIs	Parameters	Description
divisive_normalization_forward::primitive_desc()	aengine	Engine on which to perform the operation
	aprop_kind	Propagation kind. Possible values are #dnnl::prop_kind::forward_training, and #dnnl::prop_kind::forward_inference
	src_desc	Source memory descriptor
	dst_desc	Destination memory descriptor
	local_size, alpha,beta,k	Regularization local size; The alpha regularization parameter; The beta regularization parameter; The k regularization parameter
	attr	Primitive attributes (can be NULL)
	allow_empty	A flag signifying whether construction is allowed to fail without throwing an exception;
divisive_normalization_backward::primitive_desc()	aengine	Same as forward
	diff_src_desc	Diff source memory descriptor in backward
	diff_dst_desc	Diff destination memory descriptor in backward
	src_desc	Same as forward
	local_size, alpha,beta,k	Same as forward
	hint_fwd_pd	Primitive descriptor for a divisive normalization forward propagation primitive
	attr	Same as forward
	allow_empty	Same as forward

Example API Implementation

Divisive Normalization APIs:

- Affected Codes:

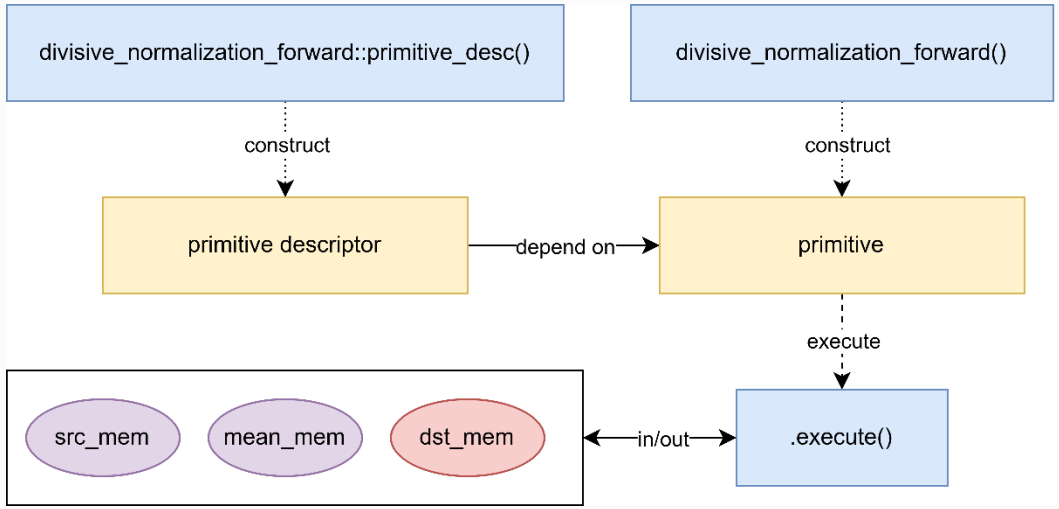
Code Files to Modify or Add	
include/oneapi/dnnl/dnnl_types.h	src/gpu/generic/sycl/ref_divisive_normalization.cpp
include/oneapi/dnnl/dnnl.h	src/gpu/generic/sycl/ref_divisive_normalization.hpp
include/oneapi/dnnl/dnnl.hpp	src/gpu/generic/sycl/sycl_primitive_conf.hpp
src/common/c_types_map.hpp	src/gpu/nvidia/cudnn_divisive_normalization_impl.hpp
src/common/divisive_normalization_pd.hpp	src/gpu/nvidia/cudnn_divisive_normalization.cpp
src/common/divisive_normalization.cpp	src/gpu/nvidia/cudnn_divisive_normalization.hpp
src/common/dnnl_traits.hpp	src/gpu/gpu_divisive_normalization_list.cpp
src/common/impl_registration.hpp	src/gpu/gpu_divisive_normalization_pd.hpp
src/common/memory_tracking.hpp	src/gpu/gpu_impl_list.cpp
src/common/opdesc.hpp	src/gpu/gpu_impl_list.hpp
src/common/primitive_desc_iface.cpp	tests/gtests/test_divisive_normalization.cpp
src/common/primitive_hashing.cpp	tests/gtests/CMakeLists.txt
src/common/primitive_hashing.hpp	examples/primitives/divisive_normalization.cpp
src/common/type_helpers.hpp	examples/CMakeLists.txt
src/common/verbose.cpp	cmake/Curses.cmake
src/common/verbose.hpp	CMakeLists.txt
src/gpu/generic/sycl/divisive_normalization_kernels.hpp	

- **Nvidia** Backend Support
- Support for a **generic** GPU, which is implemented with generic SYCL kernels.

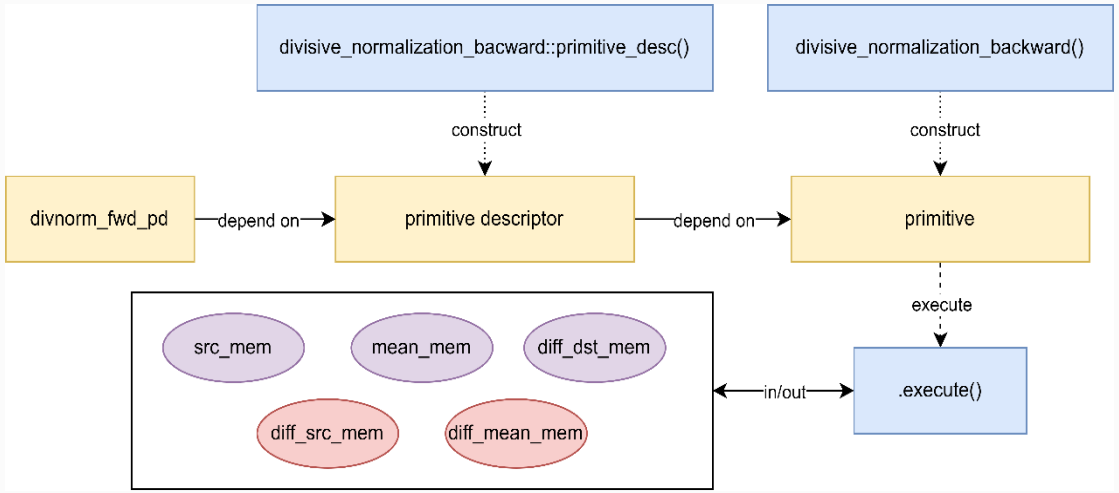
Example API Implementation

Divisive Normalization APIs:

- API Call Flow Diagram:



Forward



Backward

- Open Questions :

The workspace may not be implemented. It is a buffer that is shared between forward and backward propagation of a primitive (hence must be preserved between the calls) and is used only in training.

Q&A

Thank you for your attention!