April 2026

An overview of oneAPI Deep Neural Network Library (oneDNN)

Stefana Raileanu

Agenda

1. Introduction to oneDNN
2. Key Concepts & Basic workflow
3. Installation options
4. Debugging and profiling tools for oneDNN
5. Support
6. Q&A

Introduction to oneDNN

oneAPI Deep Neural Network Library (oneDNN)

- oneDNN is an open-source cross-platform performance library of basic building blocks for deep learning applications. oneDNN project is part of the UXL Foundation
- oneDNN is intended for deep learning applications and framework developers interested in improving application performance on CPUs and GPUs.
- Deep learning practitioners should use one of the applications enabled with oneDNN:
  - Apache SINGA
  - DeepLearning4J*
  - Flashlight*
  - llama.cpp
  - ONNX Runtime
  - OpenNMT CTranslate2
  - OpenVINO(TM) toolkit
  - PaddlePaddle*
  - PyTorch*
  - Tensorflow*

CPU optimizations

The library is optimized for the following CPUs:

- ✓ Intel® Xeon® processor E3, E5, and E7 family (formerly Sandy Bridge, Ivy Bridge, Haswell, and Broadwell)
- ✓ Intel® Xeon® Scalable processors (formerly Skylake, Cascade Lake, Cooper Lake, Ice Lake, Sapphire Rapids, and Emerald Rapids)
- ✓ Intel® Xeon® CPU Max Series (formerly Sapphire Rapids HBM)
- ✓ Intel® Xeon® 6 processors (formerly Sierra Forest and Granite Rapids)
- ✓ future Intel® Xeon® processors with Intel AVX10.2 instruction set support (code name Diamond Rapids)

- ✓ Intel® Atom® processors (at least Intel SSE4.1 support is required)
- ✓ Intel® Core™ processors (at least Intel SSE4.1 support is required)
- ✓ Intel® Core Ultra™ processors (formerly Meteor Lake, Arrow Lake, Lunar Lake, and Panther Lake)
- ✓ future Intel® Core™ processors with Intel AVX10.2 instruction set support (code name Nova Lake)

On a CPU based on Intel 64 or on AMD64 architecture, oneDNN detects the instruction set architecture (ISA) at runtime and uses just-in-time (JIT) code generation to ***deploy the code optimized for the latest supported ISA.*** *Future ISAs may have initial support in the library disabled by default and require the use of run-time controls to enable them. See CPU dispatcher control for more details.*

oneDNN 2025.3*

Features:

- API: SYCL, OpenCL, C++, C
- Training: float32, bfloat16, fp8
- Inference: float32, bfloat16, float16, fp8, int8, int4, fp4
- MLPs, CNNs (1D, 2D and 3D), RNNs, Transformers, LLMs

Support matrix:

- Compilers: Intel, GCC, CLANG, MSVC, DPC++
- OS: Linux, Windows, macOS(1) (2)
- Runtimes: SYCL, OpenCL, OpenMP, TBB, Threadpool
- CPU:
  - Intel: SSE4.1, AVX, AVX2, AVX-512, VNNI, AMX, AVX10.1, AVX10.2
  - Arm(2): AArch64, Apple Silicon, SVE 128/256/512
- GPU:
  - Intel: Xe-LP/LPG/HPG/HPC, Xe2-LPG/HPG, Xe3-LPG/XPC(2)
  - NVIDIA(2): all supported by cuDNN/cuBLAS
  - AMD(2) : all supported by MIOpen/rocBLAS

Category	Functions
Graph API	• Create, Finalize, Partition • ~80 DL operations • Optimized fused patterns (SDPA, ...)
Compute intensive operations	• Convolution/Deconvolution • Inner Product • Matmul • RNN (LSTM, LSTMP, vanilla, GRU)
Memory bandwidth limited operations	• Batch Normalization • Binary elementwise • Concat • Elementwise • Layer Normalization • Group Normalization • Local Response Normalization • Softmax • Pooling • PReLU • Resampling • Shuffle • Sum • Reduction
Data manipulation	• Reorder

(1) macOS does not have GPU support

(2) Available in open-source distribution only

* oneDNN 2025.3 in oneAPI corresponds to oneDNN open-source release v3.9.1

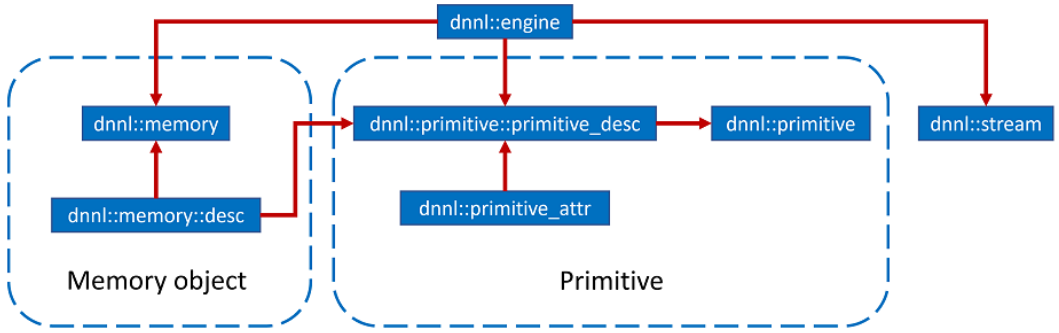
Key Concepts & Basic workflow

Primitives, engines, streams, memory objects

Key Concepts

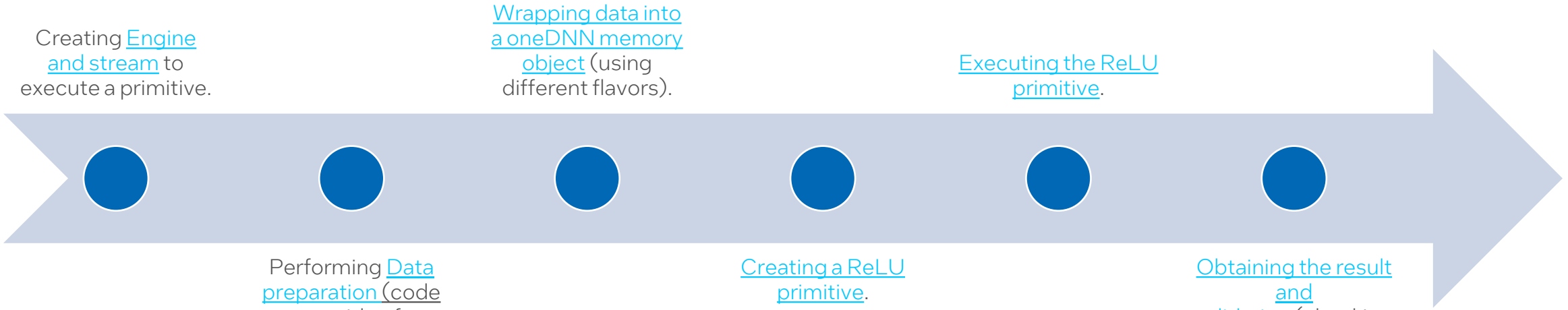
The oneDNN programming model consists in executing one or several primitives to process data in one or several memory objects. The execution is performed on an engine in the context of a stream.

oneDNN is built around the notion of a primitive (dnnl::primitive). A primitive is an object that encapsulates a particular computation such as forward convolution, backward LSTM computations, or a data transformation operation. Additionally, using primitive attributes (dnnl::primitive_attr) certain primitives can represent more complex fused computations such as a forward convolution followed by a ReLU.



- Engines (dnnl::engine) represent an abstraction of a computational device: a CPU, a specific GPU card in the system, etc. Most primitives are created to execute computations on one specific engine. The only exceptions are reorder primitives that transfer data between two different engines.
- Streams (dnnl::stream) encapsulate execution context tied to a particular engine.
- Memory objects (dnnl::memory) encapsulate handles to memory allocated on a specific engine, tensor dimensions, data type, and memory format – the way tensor indices map to offsets in linear memory space. Memory objects are passed to primitives during execution.
- **Graph extension** is a high level abstraction in oneDNN that allows you to work with a computation graph instead of individual primitives. The programming model for the graph extension is detailed in the graph basic concepts section.
- The Micro-kernel API extension (**ukernel API**) is a low-level abstraction in oneDNN that implements sequential, block-level operations. This abstraction typically allows users to implement custom operations by composing those block-level computations. Users of the ukernel API has full control of the threading and blocking logic, so they can be tailored to their application.

Basic workflow. ReLU example



Full tutorial available here.
Example code available here.

Installation

Binary & Source options

Installation options

You can download and install the oneDNN library using one of the following options:

- **Binary** Distribution: You can download pre-built binary packages from the following sources:
  - conda-forge: If the configuration you need is not available on the conda-forge channel, you can build the library using the Source Distribution.
  - Intel oneAPI:
    - Intel® oneAPI Base Toolkit
    - Intel® oneDNN standalone package
- **Source** Distribution: You can build the library from source by following the instructions on the Build from Source page.

Requirements for Building from Source

oneDNN supports systems meeting the following requirements:

- C++ compiler with C++11 standard support
- CMake 3.13 or later

The CPU engine is built by default but can be disabled at build time by setting ONEDNN_CPU_RUNTIME to NONE. In this case, GPU engine must be enabled. The CPU engine can be configured to use the OpenMP, TBB or SYCL runtime. The following additional requirements apply:

- OpenMP runtime requires C++ compiler with OpenMP 2.0 or later standard support
- TBB runtime requires Threading Building Blocks (TBB) 2017 or later.
- SYCL runtime requires
  - Intel oneAPI DPC++/C++ Compiler
  - Threading Building Blocks (TBB)

Full build instructions available here.

For the best performance results on Intel® Processors, we recommend using the Intel C++ Compiler.

Debugging and profiling tools for oneDNN

BenchDNN

OneDNN Verbose

Functionality Overview

- Built by default into oneDNN – enabled for various modes during runtime.
- Also provides a filter option which takes in a regex input and applies the verbose output to matching components:

```
ONEDNN_VERBOSE=profile_exec,filter=conv\|matmul
ONEDNN_VERBOSE=profile_exec,filter=prim
```

- Verbose mode can also be managed at runtime using the dnnl_set_verbose() function - this takes precedence over the environment variable.
- Parsing scripts are also provided as a part of oneDNN to process the verbose outputs for different use cases.

Build-Time Controls:

CMake Option	Supported Values	Description
ONEDNN_VERBOSE	**ON**, OFF	Enables verbose mode

Run-Time Controls:

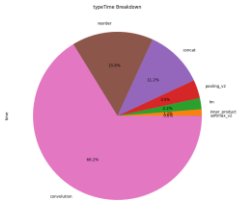
Environment Variable	Value	Description
ONEDNN_VERBOSE	none	No message printed
	error	Error messages (default)
	check	Primitive creation parameter checking information
	profile_create	Primitive creation timings
	profile_exec	Primitive execution timings
	profile	Primitive creation tracing (params+timings)
	dispatch	Primitive dispatching information
	all	Enables all above flags but none
	debuginfo=<level>	Enables internal debug printing
ONEDNN_VERBOSE_TIMESTAMP	**0**	Display timestamps disabled (default)
	1	Display timestamps enabled

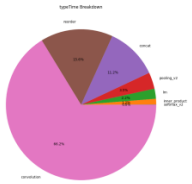
oneDNN Debugging Tools

- Performance Regression -> Extract, analyze and compare oneDNN verbose logs
  - Use *profile_utils.py* to parse oneDNN verbose logs and generate a breakdown of execution times for each primitive & kernel

softmax	0.236084
inner_product	5.540280
lrn	8.842038
pooling	26.332521
concat	47.006594
reorder	152.436031
convolution	168.416254

gemm:jit	5.540280
lrn_jit:avx512_common	8.842038
simple:any	54.040529
jit_1x1:avx512_common	63.424317
jit:avx512_common	131.560542
jit:uni	145.402096





  - Use the *verbose_converter.py* to parse a oneDNN verbose log and extract useful information for performance profiling
- BenchDNN -> a tool for robust correctness verification and performance benchmarking tool for the primitives provided by oneDNN
  - More information on how to use benchdnn: https://github.com/oneapi-src/oneDNN/blob/main/tests/benchdnn/README.md

Profile Your oneDNN Performance

with oneDNN verbose

Extract oneDNN verbose log from the framework

```
export ONEDNN_VERBOSE=1
export ONEDNN_VERBOSE_TIMESTAMP=1
```

```
onednn_verbose,v1,info,oneDNN v3.6.0 (commit 188ae7f3e3410a76a81d10b2b2bed3be7afdc307)
onednn_verbose,v1,info,cpu,runtime:OpenMP,nthr:344
onednn_verbose,v1,info,cpu,isa:Intel AVX-512 with float16, Intel DL Boost and bfloat16 support and Intel AMX with bfloat16, float16 and 8-bit integer support
onednn_verbose,v1,info,gpu,runtime:none
onednn_verbose,v1,info,graph,backend,0:dnnl_backend
onednn_verbose,v1,primitive,info,template:timestamp,operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,problem_desc,exec
_time
onednn_verbose,v1,graph,info,template:timestamp,operation,engine,partition_id,partition_kind,op_names,data_formats,logical_tensors,fpmath_mode,implementation,bac
kend,exec_time
onednn_verbose,v1,1731387421329.086914,primitive,exec,cpu,reorder,brgemm_matmul_matrix_B_reorder_t,undef,src:f32::blocked:ab::f0
dst:f32:p:blocked:BA16a64b::f0,,,3000x2000,6.84814
onednn_verbose,v1,1731387421344.190918,primitive,exec,cpu,reorder,simple:any,undef,src:f32::blocked:ab::f0 dst:f32::blocked:ab::f0,,,1000x3000,7.74609
onednn_verbose,v1,1731387421352.206055,primitive,exec,cpu,matmul,brg_matmul:avx512_core,undef,src:f32:a:blocked:ab::f0 wei:f32:ap:blocked:BA16a64b::f0
dst:f32:a:blocked:ab::f0,,,1000x3000:3000x2000,23.6719
```

Parse the verbose log to generate benchdnn inputs

```
> ./scripts/verbose_converter/verbose_converter.py -i input.log -s True
--reorder
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=abcd --dtag=aBcd8b    3x32x13x13
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=abcd --dtag=ABcd8b8a    64x32x3x3
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=aBcd8b --dtag=abcd    3x64x4x4
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=abcd --dtag=aBcd8b    3x32x13x13
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=abcde --dtag=Abcde8a    32x1x1x3x3
 --reset --allow-enum-tags-only=0 --engine=cpu     --sdt=f32 --ddt=f32  --stag=aBcd8b --dtag=abcd    3x32x4x4

--conv
 --reset --allow-enum-tags-only=0 --engine=cpu --dir=FWD_B --alg=direct --cfg=f32  --stag=aBcd8b --wtag=ABcd8b8a --dtag=aBcd8b  --attr-post-ops=eltwise_relu
 mb3_ic32oc64_ih13oh4kh3sh4dh0ph1_iw13ow4kw3sw4dw0pw1
 --reset --allow-enum-tags-only=0 --engine=cpu --dir=FWD_B --alg=direct --cfg=f32  --stag=aBcd8b --wtag=Abcde8a --dtag=aBcd8b  --attr-post-ops=eltwise_relu
 g32mb3_ic32oc32_ih13oh4kh3sh4dh0ph1_iw13ow4kw3sw4dw0pw1
```

Run benchdnn to benchmark the performance

```
./tests/benchdnn/benchdnn --matmul --engine=cpu --mode=P --fix-times-per-prb=100 --wtag=any 1000x3000:3000x2000
Output template: perf,%engine%,%impl%,%name%,%prb%,%Gops%,%+ctime%,%-time%,%-Gflops%,%0time%,%0Gflops%
perf,cpu,brg_matmul:avx512_core,,,--mode=P --fix-times-per-prb=100 --matmul 1000x3000:3000x2000,12,1.1355,1.52612,7863.06,1.74508,6876.49
tests:1 passed:1 skipped:0 mistrusted:0 unimplemented:0 invalid_arguments:0 failed:0 listed:0
total perf: min(ms):1.52612 avg(ms):1.74508
total: 0.24s; fill: 0.04s (16%);
```

Support

Questions and issues

Support

- Submit questions, feature requests, and bug reports on the GitHub issues page.
- Ticket (through LRZ)


intel