SYCLOPS

Deliverable 5.2 – SYCL deep neural network acceleration library

GRANT AGREEMENT NUMBER: 101092877

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D5.2 – SYCL deep neural network acceleration library

Executive Summary: This deliverable summarizes the achievements of work conducted under "Task 5.2: SYCL Deep Neural Network Acceleration Library" in WP5 of the SYCLOPS project. The core contribution of the SYCLOPS project was the unification and enhancement of two open source DNN libraries, namely, portDNN and oneDNN , through the SYCL standard. The final result demonstrated that the new PointNet model implementation could successfully run on different accelerators with no changes to the source code. Work done in SYCLOPS was also used in the context of a research use case involving the integration of unmanned aerial vehicles (UAVs) into shared airspace for beyond visual line of sight (BVLOS) operations, thus, demonstrating the utility of solutions developed beyond SYCLOPS. All contributions made during SYCLOPS to portDNN and oneDNN have been made available as open source in their respective Github repositories.

WP:	5		
Author(s):	Kumudha Narasimhan, Romain Biessy, Nicolo Scipione		
Editor:	Raja Appuswamy		
Leading Partner:	EUR		
Participating Partners:	CPLAY		
Version:	1.0	**Status:**	Draft
Deliverable Type:	Other	**Dissemination Level:**	PU
Official Submission Date:	30-Sep-2025	**Actual Submission Date:**	06-Oct-2025

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	EURECOM	EUR	**FR**
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	**PT**
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	**DE**
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	**CH**
5	HIRO MICRODATACENTERS B.V.	HIRO	NL
6	ACCELOM	ACC	FR
7	CODASIP S R O	CSIP	CZ
8	CODEPLAY SOFTWARE LIMITED	CPLAY	UK

Document Revision History

Version	Description	Contributions
0.1	Structure and outline	EUR
0.2	Technical contribution update	CPLAY
0.3	Revised draft post feedback	CPLAY
1.0	Final draft	EUR

Authors

Author	Partner
Romain Biessy	CPLAY
Nicolo scipione	CPLAY
Kumudha Narasimhan	CPLAY

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Stefan Roiser	CERN
Nimisha Chaturvedi	ACC
Martin Bozek	CSIP

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

Executive Summary

This deliverable summarizes the achievements of work conducted under "Task 5.2: SYCL Deep Neural Network Acceleration Library" in WP5 of the SYCLOPS project. This effort was a crucial step in advancing portable hardware acceleration for deep neural networks (DNNs), particularly addressing the requirements of autonomous systems use cases that utilize the PointNet algorithm.

The core contribution of the SYCLOPS project was the unification and enhancement of two open source DNN libraries, namely, portDNN[1] and oneDNN[2], through the SYCL standard. This strategy successfully introduced a portable path for oneDNN by developing a generic SYCL GPU backend and contributing the generic SYCL kernels originally developed for portDNN into the oneDNN ecosystem. The final result demonstrated that the new PointNet model implementation could successfully run on different accelerators with no changes to the source code. Work done in SYCLOPS was also used in the context of a research use case involving the integration of unmanned aerial vehicles (UAVs) into shared airspace for beyond visual line of sight (BVLOS) operations, thus, demonstrating the utility of solutions developed beyond SYCLOPS.

All contributions made during SYCLOPS to portDNN and oneDNN have been made available as open source in their respective Github repositories.

[1] https://github.com/codeplaysoftware/portDNN

[2] https://github.com/uxlfoundation/oneDNN

1. Introduction

Figure 1 shows the SYCLOPS hardware-software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

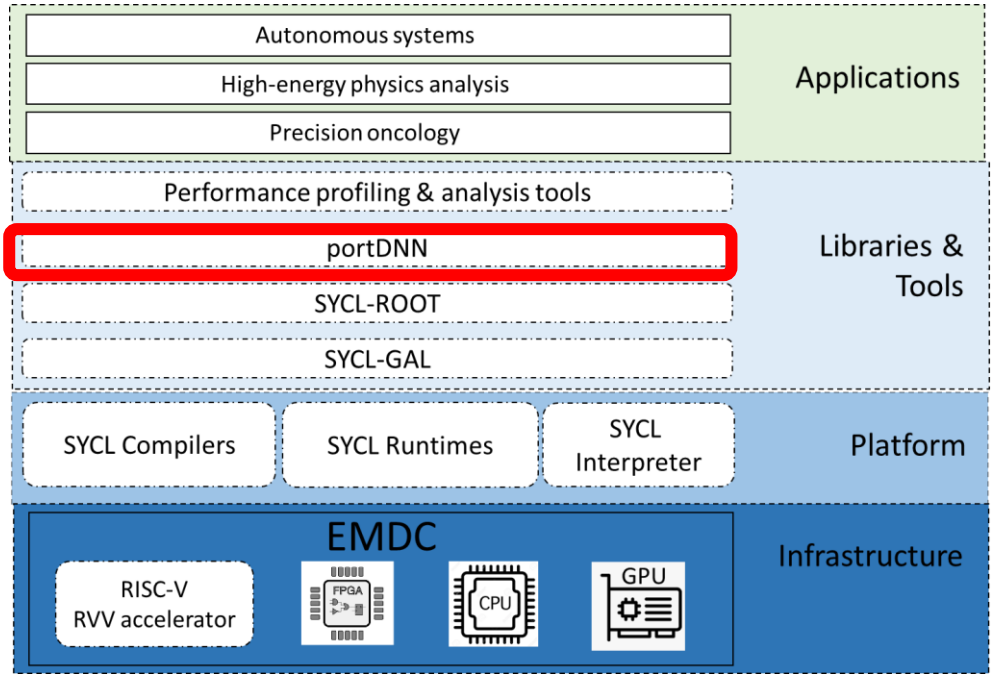

Figure 1. SYCLOPS architecture

Infrastructure layer: The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors.

Platform layer: The second layer from the bottom, the platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer. SYCLOPS will contain oneAPI DPC++ compiler from CPLAY, and AdaptiveCpp from UHEI. In terms of SYCL interpreters, SYCLOPS will contain Cling from CERN.

Application libraries and tools layer: While the platform layer described above enables direct programming in SYCL, the libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL).

This deliverable covers the **portDNN** part of the stack as highlighted in Figure 1. Several developments have been made in the context of “WP5: Task 5.2 SYCL Neural Network Library”. This deliverable is a summary of the work done.

This deliverable is structured as follows. Section 1 of this deliverable provides a high-level overview of the overall SYCLOPS architecture and positions this deliverable with respect to both components in the SYCLOPS stack and WP/tasks in the work plan. Section 2 gives the background on the autonomous systems use case of SYCLOPS and the Pointnet architecture. Section 3 describes the core work done in portDNN and oneDNN libraries to support the PointNet architecture.

2. Pointnet & Autonomous Systems

PointNet, introduced by Charles R. Qi and colleagues in 2017, represents a breakthrough in processing 3D point cloud data directly without converting it to other representations like voxel grids or images. The key innovation of PointNet lies in its ability to handle the fundamental challenge of point cloud data: permutation invariance. Unlike images where pixel positions have fixed meanings, points in a point cloud can be reordered without changing the underlying spatial information. PointNet addresses this through a clever architectural design that applies transformations to individual points before aggregating them using a symmetric function like max pooling. The model architecture is shown below.

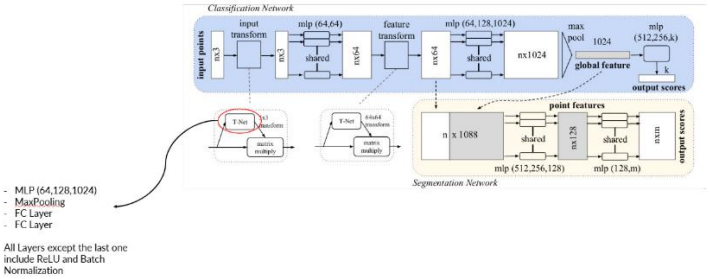

Figure 2. PointNet architecture

A major innovation of PointNet is its direct processing of unordered point clouds, a task that had previously required transforming point data into structured formats like voxel grids or multi-view images. By applying shared multilayer perceptrons (MLPs) to each point independently and then aggregating these features with a symmetric function such as max pooling, PointNet achieves permutation invariance—meaning the output remains consistent regardless of the order of input points. This makes PointNet particularly robust and efficient for various 3D recognition tasks. Furthermore, Paigwar et al. (2019) demonstrated how the architecture could be enhanced with attention mechanisms, as in their Attentional PointNet for 3D object detection in autonomous driving scenarios. By integrating channel-wise attention, their model selectively focuses on informative features, improving detection accuracy in challenging environments. While PointNet's simplicity and efficacy have inspired several follow-up architectures to better capture local geometric structures—such as PointNet++—its core approach remains foundational in the field of 3D deep learning.

PointNet and its successors have become foundational for tasks involving 3D perception, especially in autonomous driving. Modern vehicles are equipped with LiDAR and other 3D sensors, producing dense point clouds that represent the car's environment. PointNet's ability to directly process these unordered point sets allows for efficient and accurate recognition of objects such as vehicles, pedestrians, and road infrastructure. PointNet-based architectures are leveraged for real-time semantic segmentation, object detection, and scene understanding, all of which are critical for navigation and safety in autonomous systems. Their efficiency, robustness to varying point densities, and adaptability to different sensor modalities make them especially suited to the fast-paced and safety-critical demands of the automotive domain.

The autonomous systems use case in SYCLOPS led by CPLAY involves implementing and accelerating the PointNet deep learning model in an open, vendor-agnostic fashion. In doing so, the SYCLOPS project aimed to utilize PointNet to evaluate the portability and performance of its deep neural network (DNN) acceleration libraries across heterogeneous hardware.

3. portDNN and oneDNN

The SYCLOPS project addressed a core challenge in deep learning acceleration: the trade-off between high performance achieved through vendor-specific solutions and cross-platform portability. The state of portDNN and oneDNN before the project reflected this dichotomy, and the work done within SYCLOPS focused on merging their strengths to achieve performance portability.

3.1 portDNN (Formerly SYCL-DNN)

portDNN (previously known as SYCL-DNN) was an open-source deep neural network (DNN) acceleration library implemented using SYCL and C++. The core value of portDNN was providing portability via generic SYCL kernels for key deep learning primitives. It used the SYCL abstraction to allow high-performance implementations of operations like convolutions, pooling, and activations across heterogeneous hardware, including GPUs, CPUs, and FPGAs.

portDNN supported operations such as 2D convolutions (with tiled and Winograd kernels), 2D max and average pooling, and ReLU and tanh activations. Before the SYCLOPS project, while portDNN delivered portability, it only offered generic SYCL kernels for a few common operators. Crucially, to support the complex PointNet model used in the SYCLOPS autonomous driving use case, portDNN did not yet support all required operations, specifically lacking 1D convolutions, batch normalization, the concatenation operator (concat), and binary broadcast.

3.2 oneDNN (oneAPI Deep Neural Network Library)

oneDNN is an open-source library provided by the UXL Foundation that offers a unified interface for deep neural network operations. oneDNN was designed to enable developers to "write once against the oneDNN API and deploy everywhere" using the SYCL programming model. However, before the SYCLOPS project, oneDNN achieved high performance primarily through vendor-specific GPU backends, which called specific vendor libraries. This reliance on vendor-specific backends severely limited the use of oneDNN for new hardware targets. It lacked a readily deployable, portable generic path for non-traditional accelerators or new architectures.

3.3 Work done in SYCLOPS

The initial work centered on enabling PointNet functionality within the existing portable framework, portDNN.

1. *Extended Operator Support:* The project extended the operator support in portDNN to enable the full functionality of the PointNet deep learning model
2. *Added Missing Operations:* This specifically included adding support for PointNet operations such as concat, 1D convolutions, and binary broadcast
3. *Developed POC:* The original kernels and Proof of Concept (POC) for the PointNet algorithm were first developed using the portDNN library.
4. *Optimized for RISC-V:* The library was augmented with an optimized implementation of these methods, and efforts were made to optimize all operators to be able to use the RISC-V Vector (RVV) optimizations available when targeting RISC-V cores

All the work done in portDNN is open source. We have made a sample implementation of the pointnet model which can be executed with the weights available from the pre-trained model in the SYCLOPS Github organization: https://github.com/SYCLOPS-Project-EU/Pretrained-Models.

The later and defining work of SYCLOPS was the unification of the portability achieved in portDNN with the larger ecosystem of oneDNN.

1. *Extended with Generic SYCL Backend:* In collaboration with the oneDNN team, the SYCLOPS project introduced a generic SYCL GPU backend into oneDNN.
2. *Migrated Kernels:* This effort involved contributing all the generic SYCL kernels from portDNN into oneDNN.
3. *Achieved Portable Path:* This migration allowed oneDNN to gain a portable path that complemented its existing vendor-specific backends.
4. *Enabled PointNet Execution:* This integration supported the functionality of running the PointNet model through oneDNN, demonstrating the utility of oneDNN as a network to evaluate library portability across different accelerators with no code changes.

3.4 Integration

Putting it all together, we show an example oneDNN implementation in this section together with evaluation obtained using DPC++ compiler that was also developed in SYCLOPS and described in deliverable D4.2.

oneDNN is built around four core concepts:

1. Primitives: Encapsulate individual DNN operations (convolutions, pooling, activations) as reusable computational units.
2. Engines: Abstract the computational device, whether it's a CPU, GPU, or specialized accelerator.
3. Streams: Manage execution contexts and orchestrate primitive operations.
4. Memory Objects: Handle data allocation and movement across different device memory systems.

This design separates the what (operation description) from the how (execution implementation), allowing the same neural network definition to be optimized differently for each target hardware.

The first step in any oneDNN implementation involves initializing the compute device and setting up memory management. This process begins by creating a oneDNN engine that represents the computational device.

```
dnnl::engine eng(dnnl::engine::kind::gpu, 0);
dnnl::stream stream(eng);
auto sycl_queue = dnnl::sycl_interop::get_queue(stream);
```

The oneDNN library automatically detects the available hardware and configures the engine appropriately. This means your PointNet implementation can run on an Intel integrated GPU during development and seamlessly transfer to a discrete Intel or NVIDIA GPU in production. Memory management in oneDNN follows a descriptor-based approach. The example below shows how to do it for the matmul primitive used in pointnet. You define the dimensions and layout (1) of your data and create memory descriptors (2), it will create memory objects that can be efficiently accessed by different operations. The memory descriptor is passed to the primitive constructor (3). Data and execution argument (if necessary) are then set at runtime (4) without overloading primitive creation.

```
struct MMLayer : public Layer {
    MMLayer(const dnnl::engine &engine, const dnnl::stream &stream,
            const dnnl::memory &lhs_ptr, const int batch, const int m,
            const int k, const int n,
            dnnl::memory::data_type data_type = dnnl::memory::data_type::f32)
        : Layer(engine, stream), src_mem(lhs_ptr) {

        dnnl::memory::dims src_dims = {batch, m, k};
        dnnl::memory::dims weights_dims = {batch, k, n};
        dnnl::memory::dims dst_dims = {batch, m, n};

        src_desc = dnnl::memory::desc(
                src_dims, data_type, dnnl::memory::format_tag::abc);
        weights_desc = dnnl::memory::desc(
                weights_dims, data_type, dnnl::memory::format_tag::abc);
        this->out_desc_ = dnnl::memory::desc(
                dst_dims, data_type, dnnl::memory::format_tag::abc);

        this->out_mem_ = dnnl::memory(this->out_desc_, this->engine_);

        matmul_pd = dnnl::matmul::primitive_desc(
                this->engine_, src_desc, weights_desc, this->out_desc_);

        // Create Primitive
        matmul = dnnl::matmul(matmul_pd);
    }

    void execute(dnnl::memory &in_mem) override {

        // Primitive arguments.
        std::unordered_map<int, dnnl::memory> matmul_args;
        matmul_args.insert({DNNL_ARG_SRC, src_mem});
        matmul_args.insert({DNNL_ARG_WEIGHTS, in_mem});
        matmul_args.insert({DNNL_ARG_DST, this->out_mem_});

        // Primitive execution
        matmul.execute(this->stream_, matmul_args);
    }

    ~MMLayer() = default;
```

1 2 3 4

The pointnet sample using oneDNN is available at https://github.com/SYCLOPS-Project-EU/pointnet-onednn-sample. The sample was implemented to run the inference only. It has 30 layers, does about 783 kernel launches some of which use very large multi-dimensional ranges. The figure below shows the performance on an x86 CPU.

Benchmark: Pointnet (https://stanford.edu/~rqi/pointnet/)

SYCL implementation in portDNN:
https://github.com/codeplaysoftware/portDNN/tree/pointnet-sample/samples/networks/pointnet

PointNet Convolutional Neural Network Sample for 3D Pointcloud Classification [Experimental]

PointNet is a convolutional neural network architecture for applications concerning 3D recognition such as object classification and part segmentation. These sample codes implement a variant of PointNet for 3D object classification in pure SYCL code, for inference only, showing a larger example of using the portDNN compat layer. Some rough instructions for how it might be used are provided.

30 Layers

783 kernel launches:

Some with large ranges:

e.g.: [32768, 32768, 1]

Name: Intel(R) microarchitecture code named Raptorlake-DT
Frequency: 2.995 GHz
Logical CPU Count: 32
Cache Allocation Technology
Level 2 capability: not detected
Level 3 capability: not detected

WIP numbers using unmerged PRs

	NativeCPU	NativeCPU (with oneTBB)	Intel OpenCL
Elapsed time (s)	6.54	6.12	6.342
AVG Eff CPU Utilization	16 (11.65)	16.43 (12.57)	7.04

SYCLOPS

We achieved competitive performance on x86 (RaptorLake-DT) where NativeCPU using the oneTBB backend that executes the network in 6.12s while Intel SYCL+OpenCL takes 6.342s. Using the default backend (without oneTBB) executes the network 6.54s, which highlights the performance benefit of oneTBB on that platform. In addition the CPU, we have also tested this sample on Nvidia A4000 and Intel BMG GPU.

3.5 Research use case of portDNN and oneDNN

While the SYCLOPS project focused on optimizing and accelerating the resultant model using portDNN and oneDNN for inference, we collaborated with researchers from the European Commission funded RAPID project (Risk-aware Automated Port Inspection Drone) on their work on using airborne radar technology for the end-to-end semantic segmentation of aerial point clouds. Detailed information about this work is available in the associated peer-reviewed publication[3]. We provide an overview here to highlight work the work on portDNN in SYCLOPS benefit this research.

The research addresses the need for enhanced situational awareness in UAVs to enable safe and efficient Beyond Visual Line of Sight (BVLOS) operations. Current approaches often target single object detection or rely on simpler sensing outputs, lacking the rapid end-to-end processing necessary for safety-critical insights. The study focuses on multi-object point cloud segmentation of collision hazards to improve UAV situational awareness.

The framework adapts and optimizes the PointNet architecture for aerial contexts, integrating insights specific to the aerial domain. The study utilizes airborne radar technology, specifically an Oculii EAGLE 4D imaging/multiple-input multiple-output (MIMO), 76-81GHz W-band radar designed for automotive use but applied here to an air-to-air use case. The system leverages point clouds, which capture the 3D spatial distribution of radar returns and provide richer spatial information compared to radar spectrograms, and are also considered a more efficient data representation. The framework successfully distinguishes five unique classes commonly expected in complex aerial scenes: (i) large mobile drones (DJI M300 RTK), (ii) Small mobile drones (DJI Mini), (iii) airplanes (Ikarus C42), (iv) static returns (ground), (v) static returns (infrastructure).

The work involved a significant focus on network optimization and acceleration, addressing the stringent SWaP (size, weight, and power) constraints inherent to UAV operations. CPLAY assisted the researchers by starting with an un-optimized PointNet-SYCL model and used portDNN to refine the implementation of several core operations within the PointNet architecture (Concatenation, Broadcasted Binary Operations, Softmax, and Pooling). A key optimization achieved using portDNN was the transformation of 1D Convolutions into Matrix Multiplications, which effectively reduced computation time. Leveraging SYCL via portDNN made it possible to optimize PointNet operations (like convolutions, fully connected layers, pooling, and activations) to align with aerial use case needs while abstracting underlying hardware specifics.

[3] https://ieeexplore.ieee.org/document/10660655/

4. Conclusion

This deliverable concludes the work conducted on portDNN and oneDNN in the context of “Task 5.2: SYCL deep neural network acceleration library” in WP5 within the SYCLOPS project, and was a significant effort aimed at advancing portable hardware acceleration for deep neural networks (DNNs), particularly for autonomous systems use cases involving the PointNet algorithm.

The project addressed limitations in existing DNN acceleration solutions. oneDNN (oneAPI Deep Neural Network Library) previously provided high performance but relied on vendor-specific GPU backends, which restricted its applicability to new hardware targets. portDNN (formerly SYCL-DNN), an open-source SYCL/C++ library for accelerating neural network algorithms, offered portability through generic SYCL kernels but only for a few common operators. The contributions of the SYCLOPS project focused on unifying and enhancing these two libraries using the SYCL standard.

The integration provided oneDNN with a portable path that complemented its existing vendor-specific backends, while ensuring the continued efficiency of the work initially done in portDNN within a larger ecosystem. The result demonstrated that the new PointNet model implementation can successfully run on different accelerators with no change to the code, serving as a viable network for evaluating the portability of the library. The inference-only sample of PointNet using this infrastructure was tested on devices including an Nvidia A4000 and an Intel BMG GPU.