Template-Based Deployment for Real-Time Deep Learning on Point Clouds

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

M.Sc. Marc Neu

geboren in Leonberg

Tag der mündlichen Prüfung:	28.08.2026
Hauptreferent:	Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent:	Prof. Dr. Torben Ferber

Template-Based Deployment for
Real-Time Deep Learning on Point Clouds

First edition: September 2026

DOI: 10.5445/IR/1000196764

This page intentionally left blank

This page intentionally left blank

Disclaimer

Algorithm development and data analysis in high-energy physics, such as the work presented in this doctoral thesis, are collaborative efforts. The SuperKEKB particle accelerator, which provides the particle beams essential for all studies at Belle II, was built and is operated and maintained by the SuperKEKB accelerator group. The Belle II detector was built and is maintained and operated by the Belle II collaboration. The Belle II collaboration also creates the centrally provided simulated and recorded datasets and maintains the computing infrastructure necessary to process them. The software environment necessary for studies with Belle II data plays an important role and was created and is maintained by the collaboration. I have been a part of the Belle II collaboration since 2023 and performed all studies detailed in this thesis except for the following:

- The algorithm for the Belle II hit cleanup was developed by Greta Heine. The pretrained, quantised model was provided to me as an input for my case study.
- The algorithm for the Belle II calorimeter clustering, called CaloClusterNet, was developed by Isabel Haide. The pretrained, quantised model was provided to me as an input for my case study.
- The CaloClusterNet model, later called Sunset, was trained by Frank Baptist.
- The CaloClusterNet model, later called Long Shutdown 2, was trained by Thomas Lobmaier.

This thesis uses Artificial Intelligence tools to assist with grammatical and stylistic improvement of text and with the creation of program code. Claude[1] is utilised throughout the thesis for spell and grammar checks, as well as for paraphrasing individual, selected sentences to improve clarity and precision in academic writing. The model versions used are Claude Opus 4.6–4.8 and Claude Sonnet 4.6. Where Artificial Intelligence tools were used to assist with program code, all generated code was reviewed and tested by me. I have approved all suggested changes and take full responsibility for the content of this thesis.

[1] https://www.anthropic.com/claude (Accessed throughout 2025–2026)

This page intentionally left blank

Abstract

Deep learning on point clouds is well suited to irregular detector geometries with millions of individual sensors, yet no deployment methodology exists for real-time data acquisition and trigger systems. This thesis presents DeePloy, a generalisable methodology for deploying graph-based Point Cloud Networks onto heterogeneous, commercially available reprogrammable hardware under hard real-time constraints. These range from Field-Programmable Gate Array devices to Systems on Chip that combine Field-Programmable Gate Arrays with Coarse-Grained Reconfigurable Arrays. Point Cloud Networks are decomposed into a graph of operators mapped onto a dataflow accelerator architecture, with topology-dependent transformations proposed to improve inference latency and throughput. The methodology is validated through three case studies in the Belle II experiment. First, a Point Cloud Network for the Central Drift Chamber hardware trigger is deployed on the AMD Alveo V80, achieving 0.425 µs latency and 32 million detector snapshots per second in out-of-context simulation, an almost ten-fold improvement in throughput over prior work. Second, a Point Cloud Network for the Electromagnetic Calorimeter hardware trigger is developed and integrated into the trigger system, operating in Belle II since early 2026, meeting an end-to-end latency deadline of 1.053 µs and processing 8 million detector snapshots per second on the AMD Ultrascale XCVU190. This approach constitutes the first deployment of a Point Cloud Network-based reconstruction algorithm in a particle physics hardware trigger. Third, a Point Cloud Network is deployed on the AMD Versal VCK190, achieving a 7.02× throughput speedup over a Graphics Processing Unit baseline in hardware measurements. Collectively, this work demonstrates that graph-based Point Cloud Networks can meet the hard real-time demands of hardware trigger systems, enabling reconstruction algorithms in large-scale scientific experiments.

This page intentionally left blank

Kurzfassung

Deep Learning auf Punktwolken eignet sich für irreguläre Detektorgeometrien mit Millionen einzelner Sensoren, jedoch existiert bislang keine Methodik für dessen Einsatz in Echtzeit-Datenakquisitions- und Triggersystemen. Diese Dissertation stellt DeePloy vor, eine generalisierbare Methodik zur Abbildung graphbasierter Punktwolkennetzwerke auf heterogene, kommerziell verfügbare rekonfigurierbare Hardware unter harten Echtzeitanforderungen. Diese Plattformen reichen von Field-Programmable Gate Arrays bis hin zu Systems on Chip, die Field-Programmable Gate Arrays mit Coarse-Grained Reconfigurable Arrays kombinieren. Punktwolkennetzwerke werden in einen Graphen bestehend aus Operatoren zerlegt und auf eine Datenfluss-Architektur abgebildet, wobei topologieabhängige Transformationen zur Verbesserung von Latenz und Durchsatz der Inferenz vorgeschlagen werden. Die Methodik wird anhand von drei Fallstudien im Belle II Experiment validiert. Erstens wird ein Punktwolkennetzwerk für den Hardware-Trigger der Central Drift Chamber auf dem AMD Alveo V80 implementiert und erreicht in Simulationen eine Latenz von 0,425 µs sowie 32 Millionen Detektoraufnahmen pro Sekunde, was einer nahezu zehnfachen Durchsatzsteigerung gegenüber vorherigen Arbeiten entspricht. Zweitens wird ein Punktwolkennetzwerk für den Hardware-Trigger des elektromagnetischen Kalorimeters entwickelt und in das Triggersystem integriert, das seit Anfang 2026 in Belle II in Betrieb ist, eine Ende-zu-Ende-Latenzvorgabe von 1,053 µs einhält und 8 Millionen Detektoraufnahmen pro Sekunde auf dem AMD Ultrascale XCVU190 verarbeitet. Dieser Ansatz stellt die erste Implementierung eines auf Punktwolkennetzwerken basierenden Rekonstruktionsalgorithmus in einem Hardware-Trigger in der Teilchenphysik dar. Drittens wird ein Punktwolkennetzwerk auf dem AMD Versal VCK190 implementiert und erreicht in Hardwaremessungen eine 7,02× höhere Durchsatzleistung gegenüber einer Grafikkarte. Zusammenfassend zeigt diese Arbeit, dass graphbasierte Punktwolkennetzwerke die harten Echtzeitanforderungen von Hardware-Triggersystemen erfüllen können und damit Rekonstruktionsalgorithmen in wissenschaftlichen Großexperimenten realisiert weden können.

This page intentionally left blank

Contents

Notation

This chapter introduces the notation and symbol types which are used in this thesis.

General notation

Numbers	1.0	Numbers are set like this: 1.0.
Code	`code`	Inline program code is set in `typewriter` font.
Identifiers	`lml7`	Names defined in documentation are reproduced verbatim.
Scalars	a	Scalars are set in lowercase.
Vectors	$\vec{a}$	Vectors are set with overline arrows.
Matrices	A	Matrices are set in uppercase.
Sets	$\mathcal{A}$	Sets are set in calligraphic style.
References	[1]	References are set like this.
Own works	Neu23	Own works are set like this.

This page intentionally left blank

Chapter 1

Introduction

This thesis addresses deploying Point Cloud Networks (PCNs) on commercially available reprogrammable hardware by developing a dedicated deployment methodology. This work is motivated by the technical requirements for algorithm deployment in multinational research infrastructure, such as particle detectors and accelerators. The methodology presented in this thesis was developed during the operation of the Belle II particle detector in Tsukuba, Japan. It has since been generalised and further refined as part of the DEEP research project. While this thesis provides a brief overview of state-of-the-art algorithms relevant to large-scale scientific experiments, the reader is referred to closely related PhD theses on algorithm development for the Belle II experiment [1, 2].

Research infrastructures generate ever-growing amounts of data to advance fundamental research into matter and the universe. Large-scale scientific experiments operate continuously around the clock for years to decades to aggregate as much data as possible, imposing stringent technical requirements on the Data Acquisition (DAQ) systems. Figure 1.1 depicts an overview of the throughput and latency requirements of four selected current experiments and one commercial system with similar requirements. If possible, a differentiation is made between hardware and software triggers. Hardware triggers are typically deployed on custom hardware architecture developed specifically for the experiment, achieving hard real-time characteristics. Software triggers typically run on one or more high-performance compute clusters, achieving soft real-time characteristics.

The figure shows two experiments, DUNE [3] and IceCube [4], with medium-throughput requirements ranging from 10 GB/s to 1 TB/s in the bottom centre. Similar requirements are imposed on the software trigger at Belle II [5] at the SuperKEKB facility [6]. More demanding are the software triggers of the Compact Muon Solenoid (CMS) experiment [7] and the ATLAS experiment [8] at the Large Hadron Collider (LHC) at European Organization for Nuclear Research (CERN), which process more than 100 GB/s at latencies between 10 ms and 1 s. The most demanding systems, however, are hardware trigger systems, which typically serve as

the first data-filtering stage in the DAQ pipeline. Both hardware triggers, from experiments at CERN and at the Belle II experiment, process between 10 TB/s to 1 PB/s [9, 10, Lai25]. Latency requirements differ slightly from 5 µs for the current Belle II hardware trigger to 12.5 µs for hardware triggers at CERN after the Phase-2 trigger upgrade.

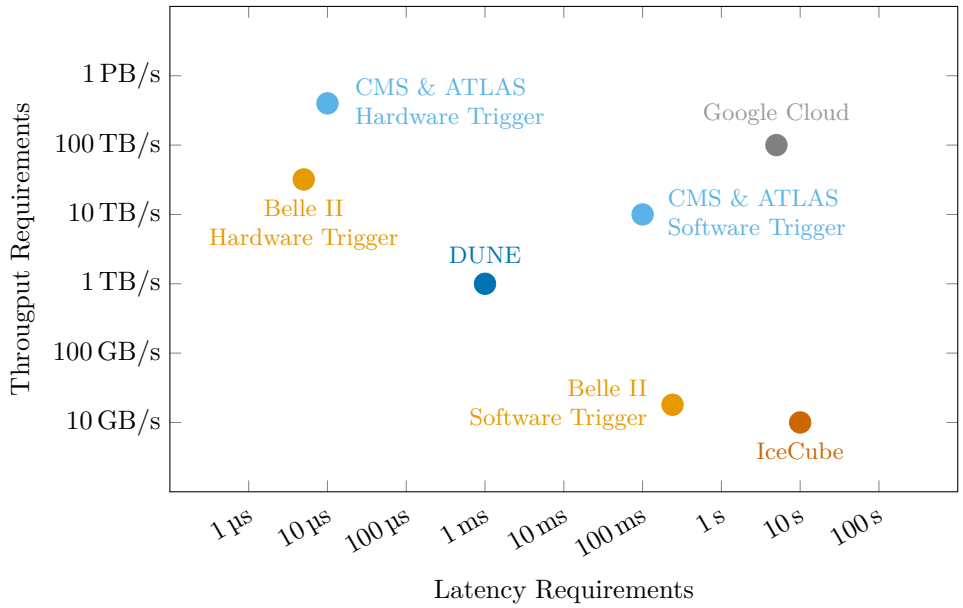


Figure 1.1: Approximate latency and throughput requirements for selected large-scale scientific experiments. Values for LHC, DUNE, IceCube, and Google Cloud are taken from ref. [11]. Values for Belle II are based on the experiment status in 2026.

These requirements directly shape the architecture of DAQ systems in such experiments. To give a better overview of how these technical requirements influence the design of DAQ systems for large-scale scientific experiments, a typical data processing pipeline is shown in figure 1.2. The figure highlights two types of modules. **Soft real-time modules** on the readout path are shown in blue: sensors, preprocessors, and software triggers. For the operation of the experiments, these modules must maintain high throughput and reliability. **Hard real-time modules** are highlighted in green: machine control, hardware trigger, and calibration. Compared with soft real-time modules, these modules give direct feedback to the DAQ system. In the following, these modules will be explained in more detail with real-world examples.

Machine control involves dynamically optimising accelerator parameters to ensure efficient experiment operation and beam uptime. Due to the low latency required for accurate real-time control, such systems are implemented either on a full-custom Application Specific Integrated Circuits (ASICs) or on Field Programmable Gate Arrays (FPGAs). Machine control

measures are planned for application to superconducting radio-frequency cavities at the European XFEL [12] and PETRA IV [13].

Hardware triggers are hard real-time systems which implement reconstruction algorithms within a 1 µs to 10 µs latency budget. The goal of hardware triggers is to identify relevant events from a continuous data stream delivered by the frontend electronics, thereby reducing the computational load on downstream stages in the DAQ system. As previously described, such hardware triggers are employed in the CMS and ATLAS experiments at the LHC, the Belle II experiment, and in smaller experiments such as Amber [14].

Calibration of the recorded data is performed to optimise accuracy and consistency under the experimental conditions. Traditionally, calibration is carried out on persistent offline data. However, moving the calibration step to the online processing domain inside the DAQ system reduces the overall data size. Such an application is under investigation at the LHCb experiment [15] at CERN for the alignment of tracking sensors in the detector.

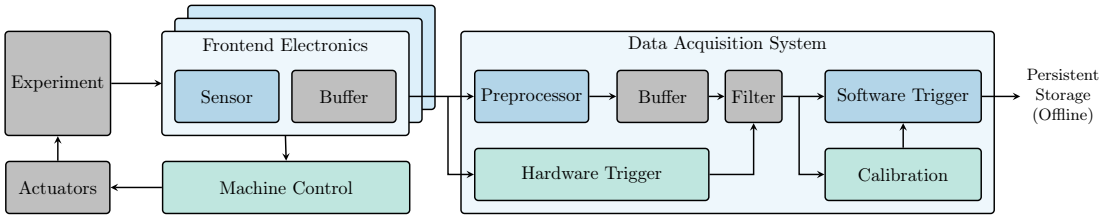


Figure 1.2: Typical data processing pipeline in large-scale scientific experiments. Soft real-time modules are highlighted in green, modules that directly interact with the DAQ system are highlighted in blue, and the remaining modules are highlighted in grey.

As these requirements span several orders of magnitude in both throughput and latency, the selection of an appropriate compute platform is a determining factor in whether a DAQ system can fulfil its operational requirements. As multinational experiments are planned decades in advance, the hardware platforms selected for deploying the DAQ system must be chosen carefully.

Figure 1.3 shows a tradeoff between performance and flexibility for common compute platforms. Central Processing Units (CPUs) is the most commonly available and flexible platform today. Best suited for sequential tasks, CPUs can execute any program, at the cost of comparatively lower performance and energy efficiency. Graphics Processing Units (GPUs), originally designed for video processing, have found wide adoption in scientific and commercial applications where algorithms are parallelisable. Neural Processing Units

(NPUs), which can be seen as an application-specific specialisation of GPUs, offer better performance for tasks such as machine learning, or digital signal processing. FPGAs are programmable devices capable of implementing arbitrary synchronous digital circuits even after fabrication. Interested readers may find a more thorough description in section 2.3.1. Coarse Grain Reconfigurable Arrays (CGRAs) are, like FPGAs, considered reprogrammable devices. In comparison, they consist of larger, reconfigurable tiles, thereby achieving better area efficiency than fine-grained reconfigurable architectures. Interested readers may find a more thorough description in section 2.3.2.

Generally, the relative performance improvement of custom circuits such as CGRAs and FPGAs comes at the cost of flexibility and programmability. It stands to reason that the time to market for products built on these platforms is longer, and implementations are more error-prone, requiring more testing and validation than those of general-purpose counterparts. As a result, the deployment of algorithms onto reprogrammable hardware platforms is an active area of research.

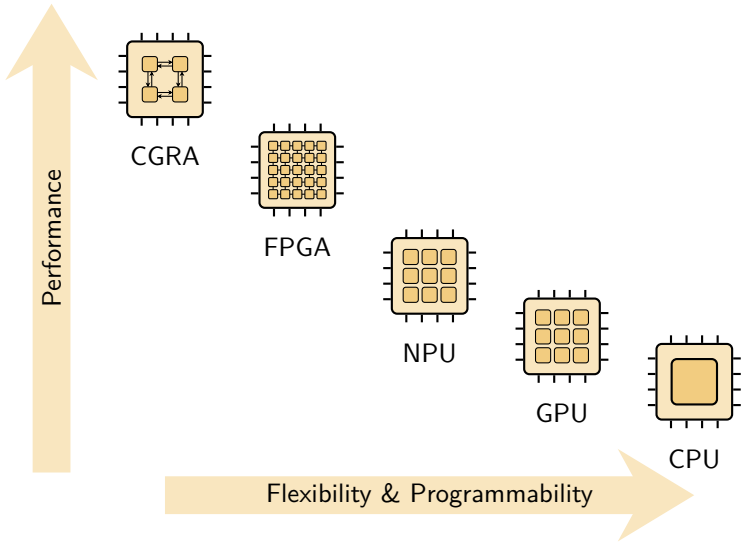


Figure 1.3: Performance vs flexibility tradeoff for common compute platforms. Adapted from ref. [16].

In recent years, machine learning techniques have been explored for non real-time reconstruction in high-energy physics across a broad range of tasks, including track reconstruction [17], hit filtering [Hei26b, Hei26c], clustering of energy depositions [18, 19], and anomaly detection [20]. Due to the irregular geometry and variety of sensors used in large-scale scientific experiments, reconstruction systems benefit from the relational structure modelled by PCNs [21, 22, 23]. These architectures operate directly on graphs and point clouds, re-

spectively, and unlike more conventional networks such as Convolutional Neural Networks (CNNs), they explicitly account for the unstructured and unordered nature of the input data, enabling improved feature extraction. A more thorough introduction to deep learning on point clouds is provided in section 2.2.

Despite their strong representational capabilities, PCNs are well known for their substantial computational cost and limited computational efficiency. Both factors arise from the high degree of irregularity and sparsity inherent in graph and point cloud data, which result in dynamic execution behaviour that is difficult to map efficiently onto high-performance platforms such as FPGAs and CGRAs hardware accelerators. Combined with the strict latency and throughput constraints of hardware trigger systems in large-scale scientific experiments, these properties make the real-time deployment of Graph Neural Networks (GNNs) and PCNs a non-trivial problem.

This thesis addresses how PCNs can be efficiently deployed on reprogrammable hardware platforms to meet the real-time requirements of large-scale scientific experiments. To address this challenge, chapter 4 describes the end-to-end deployment methodology DeePloy, developed with a focus on the practical needs of engineers in high-energy physics. The methodology is evaluated through three case studies presented in this thesis. Chapter 5 describes the deployment of a static PCN-based hit filtering algorithm for the Belle II experiment in an experimental study. Chapter 6 describes the deployment of a dynamic PCN-based clustering algorithm for the Belle II experiment, validated during operation in 2024, 2025, and 2026. Chapter 7 describes the extension of this work to heterogeneous System-on-Chips (SoCs), targeting future upgrades of the Belle II experiment as well as other particle detectors of similar scale, such as the CMS detector at the LHC at CERN.

As part of this thesis, various new topics have been investigated or advanced significantly beyond the state of the art. The following list provides an overview and references the relevant section in the dissertation.

- Introduction of DeePloy, an end-to-end deployment methodology unifying the mapping schemes, architecture template library, and hardware modules described below into a single toolflow (chapter 4).
- Introduction of a deployment methodology for static PCNs on FPGAs (section 4.3).
- Introduction of a deployment methodology for dynamic PCNs (section 4.4), extending the message passing programming model for GNNs and PCNs with a formulation for dynamic graph building.

- Development and realisation of an architecture template library to facilitate the previously mentioned mapping schemes on FPGAs and on CGRAs (section 4.5).
- Development of a real-time sparsity compression hardware module for FPGAs, optimized for low-latency environments (section 4.6).
- Conceptualization and deployment of a static PCN-based hardware trigger on FPGA for hit filtering for the Belle II experiment (chapter 5).
- Deployment, commissioning, and operation of a dynamic PCN-based hardware trigger on FPGA. This system is the first of its kind operational in a particle physics experiment (chapter 6).
- Optimization, partitioning, and deployment of a dynamic PCN on heterogeneous SoCs, combining FPGA and CGRA partitions (chapter 7).

Chapter 2

Background

This chapter summarises the relevant basics. It covers an overview of the Belle II Experiment in section 2.1, a review of PCNs in section 2.2, as well as an overview of relevant deployment targets in section 2.3.

2.1 Belle II Experiment

The Belle II Experiment [5] at the SuperKEKB electron-positron collider [6] in Tsukuba, Japan, studies flavour and dark sector physics at the Y(4S) resonance energy of 10.58 GeV. This operation point allows the accelerator to produce B-mesons in large quantities [24]. The double-ring structure of the accelerator features asymmetric beam energies: 4.00 GeV for the low-energy ring and 7.01 GeV for the high-energy ring. The design luminosity of $6 \times 10^{35}\ \mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ will be achieved through the Nano-Beam scheme, originally developed for the Italian Super B factory [25]. As of December 2024, the SuperKEKB accelerator has reached a world-record instantaneous luminosity of $5.1 \times 10^{34}\ \mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ [26].

The Belle II Detector is the successor to the previously designed Belle Detector [27] at the SuperKEKB facility. Its main concern is to maintain Belle's current performance in an environment with considerably higher background levels [5]. As a hermetic detector, it consists of several subdetectors oriented around the interaction point, which is the origin of the lab frame (x,y,z). The z-axis points approximately in the direction towards the electron beam. The x-axis aligns horizontally, and the y-axis aligns vertically in the right-hand coordinate system. The polar angle $\theta \in [0,\pi]$ is defined as $\theta = \arccos\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right)$. The azimuthal angle $\phi \in (-\pi,\pi]$ is defined as $\phi = \mathrm{atan2}(y,x)$. To force charged particles onto a curved trajectory, tracking detectors are enclosed in a magnetic field of approximately 1.5 T. An overview of the detector is shown in figure 2.1.

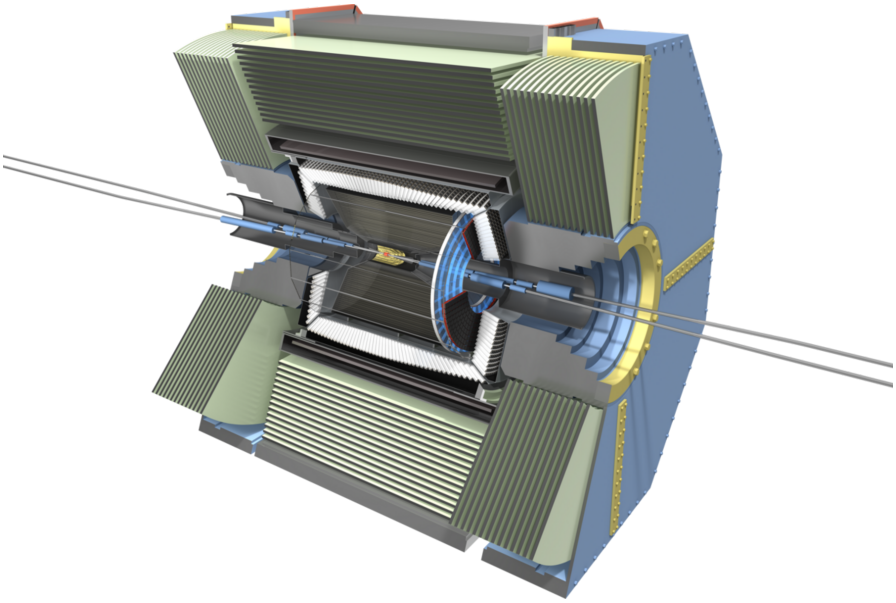


Figure 2.1: Three-dimensional model of the Belle II detector. The interaction point is in the centre of the detector. Figure taken from ref. [28].

In total, the detector consists of seven subdetectors. From inside to outside these are namely: Pixel Detector (PXD), Silicon Vertex Detector (SVD), Central Drift Chamber (CDC), Time-Of-Propagation Counter (TOP), Aerogel RICH Detector (ARICH), Electromagnetic Calorimeter (ECL), and K_L^0 / Muon Detector (KLM). In the following, I will give a short overview of all subdetectors.

The Pixel Detector is the subdetector at Belle II that is closest to the interaction point [29]. It is based on depleted p-channel field-effect transistors, with each transistor representing a single pixel in the sensor array. Each module contains 192 k pixels across an active area of 12.75 cm^2, enabling fine-grained positional resolution.

The Silicon Vertex Detector is the second vertex detector at Belle II. It is composed of four layers of double-sided silicon strip detectors [30]. Its role is to track low-momentum charged particle tracks and reconstruct $K_S^0 \to \pi^+\pi^-$. Additionally, it supports extrapolation from the Pixel Detector (PXD) to Central Drift Chamber (CDC).

The Central Drift Chamber is responsible for sensing charged particles leaving the interaction point, which propagate outwards on a helix [31]. Another important task is particle identification by measuring the energy loss per unit length, $\frac{\delta E}{\delta x}$. Its main component is an enclosed gas chamber filled with a 50% helium, 50% ethane mixture. A more thorough description of the CDC will be given in section 2.1.2.

The Time-Of-Propagation Counter is positioned radially between the tracking detector and the Electromagnetic Calorimeter (ECL), measuring the propagation time and hit positions of Cherenkov photons to identify pions, kaons, protons, and deuterons [32]. When charged particles cross polished bars of synthetic fused silica, they radiate Cherenkov photons, which are measured by an array of photomultiplier tubes.

The Aerogel RICH Detector is used in the forward endcap region to separate kaons from pions with momenta up to 4 GeV/c [33]. It consists of a silica aerogel radiator and hybrid avalanche photo detectors.

The Electromagnetic Calorimeter measures the energy depositions of neutral and charged particles using 8736 thallium-doped cesium iodide (CsI(Tl)) crystals [34]. The crystals themselves are kept unchanged from the first Belle experiment. A more thorough description of the ECL will be given in section 2.1.4.

The K_L^0 / Muon Detector is the outermost subdetector at Belle II and is composed of 41 active particle detection modules [35]. Depending on the module's position, either resistive-plate counters or plastic scintillators are used. The K_L^0 / Muon Detector (KLM) detects charged particles either by amplifying electrical signals or by scintillation light generated in the detector [36].

2.1.1 First-Level Trigger System

The Belle II DAQ System [37] receives all sensor data from the detector frontend electronics via the Belle2Link [38], and the COmmon Pipelined Platform for Electronics Readout (COOPER) [39] combines these separate, parallel data streams into global aggregations. The aggregated snapshot of all sensor data from a subdetector in a given sampling period is referred to as *data window*. Because data windows only cover a short timespan, for example 125 ns, a sliding window is placed over adjacent data windows to avoid losing collisions at the boundary of the sampling period. On trigger level, multiple adjacent data windows are combined into trigger windows. *trigger windows* cover one or more data windows and often have a reduced precision compared to the full offline resolution. For offline analysis, all data windows over an extended time span are combined into an *event*. Due to the large volume of raw data intended for later physics analysis, the readout rate of the DAQ System is limited to 30 kHz [37]. In the literature, this readout rate is often referred to as a unitless throughput. In this work, the throughput is typically referred to as detector snapshots per second, to avoid confusion between data windows, trigger windows, and events.

Without further preprocessing, the readout rate would exceed its design limit during operation. Thus, a two-stage online data reduction scheme is applied. First, the First-Level Trigger System (L1 Trigger) [40] is an FPGA-based system that reduces the readout rate to the maximum design rate of 30 kHz. Second, the High-Level Trigger System (HLT) is a software-based system deployed on a general-purpose CPUs that runs the full reconstruction using the Belle II Analysis Framework to further reduce the readout rate before storage on hard drives. For persistent storage on hard drives, the DAQ system supports a maximum readout rate of 10 kHz. In the following, the First-Level Trigger System (L1 Trigger) will be explained in more detail.

The L1 Trigger consists of several FPGA-based platforms connected via high-speed gigabit transreceivers (GTs) for high-throughput, low-latency data transmission. The purpose of the L1 Trigger is to generate a binary signal for the DAQ System, indicating that the current detector snapshots in the readout buffers should be retained for further processing. Because the L1 Trigger system employs strict in-order processing and the readout buffers in the detector frontend electronics are memory-limited, it is a hard real-time system with a latency requirement of 5 µs. At the same time, it must operate deadtime-free, at a design throughput of 32 MHz[1]. table 2.1 shows the practical latency limits for the L1 Trigger based on readout buffer limits of respective subdetectors.

Table 2.1: Estimated maximum latency for the First-Level Trigger System (L1 Trigger) for different subdetectors at Belle II. Taken from ref. [1]

Subdetector	Max. Latency
Pixel Detector	10 µs
Silicon Vertex Detector	5 µs
Central Drift Chamber	15 µs
Time-Of-Propagation Counter	9 µs
Aerogel RICH Detector	1015 µs
Electromagnetic Calorimeter	100 µs
K_L^0 / Muon Detector	5.2 µs

An overview of the dataflow inside the L1 Trigger is shown in figure 2.2. Subdetectors that participate in the first-level trigger decision are the CDC, the ECL, the Time-Of-Propagation Counter (TOP), and the KLM.

The CDC trigger system provides tracking information for charged particles, namely the momentum, the offset from the interaction point along the z-axis, and the ϕ angle. A more

[1] The L1 Trigger system generates a global trigger decision every 32 MHz. Individual subtriggers may realise a lower throughput.

thorough explanation of this system is given in section 2.1.3.

The ECL trigger system provides cluster information based on energy depositions of charged particles. A more thorough explanation of this system is given in section 2.1.5.

The TOP trigger system provides additional timing information and hit information based on its sensor input.

The KLM trigger system provides additional cluster information for minimum-ionizing particles like muons.

Tracking information for the CDC subtrigger is matched against cluster information from the ECL in the Global Reconstruction Logic (GRL) to obtain a global view [Lai25]. The Global Decision Logic (GDL) receives information from all subtriggers and the Global Reconstruction Logic (GRL) to form a global trigger decision, which signals the DAQ system to read out all subdetectors. Information transmitted to the GDL is provided through a set of Boolean flags. These flags, referred to as trigger bits, classify the current detector snapshot based on certain conditions. As an example, the two-cluster trigger bit on Isolated Cluster Network Electromagnetic Calorimeter Trigger Module (ICN-ETM) is true if at least two clusters in the inner region of the ECL detector have been found with a per-cluster threshold of at least 100 MeV. In general, trigger bits pass through three submodules: First, the input trigger delay submodule checks the connection to the GDL for liveness and measures the subtrigger latency via the `active` signal. Variable-length shift registers are used to synchronise all incoming subtriggers based on the measured delay. Second, an optional veto (bitwise `AND` with the inverted veto signal) is applied to the input trigger signals. Two vetoes used for the Graph Neural Network Electromagnetic Calorimeter Trigger Module (GNN-ETM) trigger signals are the injection veto and the Bhabha veto. The injection veto suppresses beam-induced background. The Bhabha veto suppresses the high-rate Bhabha scattering process. Without applying these vetoes, both would dominate the resulting trigger rates. Third, a prescale and mask value is applied to all trigger bits. This effectively enables the run operators to disable single bits (masking) or to reduce the trigger rate of these bits by triggering only on the nth occurrence (where n is the prescale factor). Based on the prescaled trigger bits, a global trigger decision is realised. The GDL realises a global trigger decision by merging prescaled trigger bits using a `OR`-reduce Boolean equation, and subsequently forwards the global decision to the Frontend Timing Switch. A minimum hold-off time of 200 ns between two subsequent L1 Trigger signals is enforced [35].

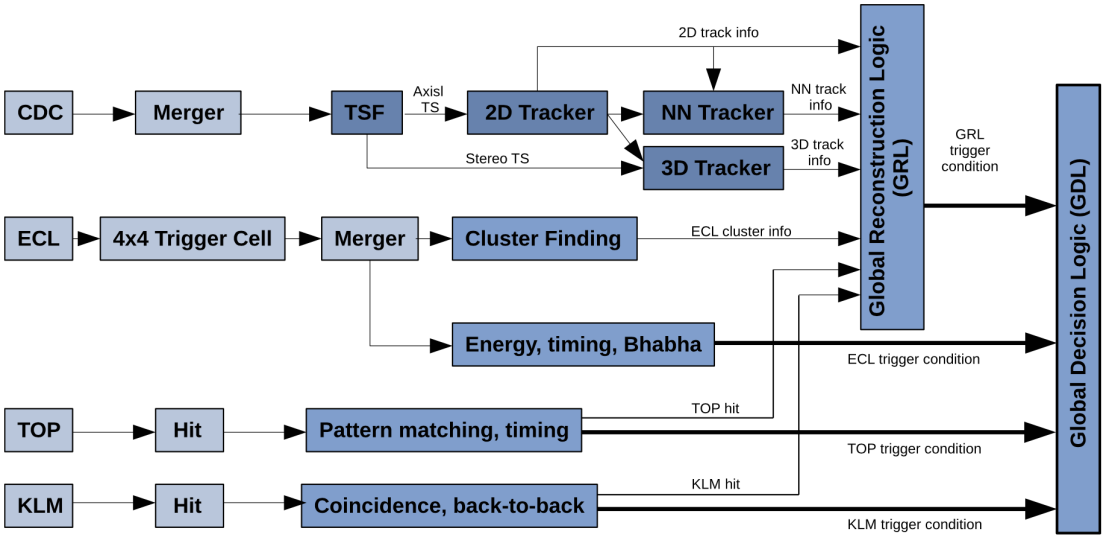


Figure 2.2: Block diagram of the First-Level Trigger System (L1 Trigger), indicating dataflow and logical system architecture [Lai25].

The individual stages of the L1 Trigger are deployed on so-called Universal Trigger Board (UT) boards. The third generation of the UT boards (UT 3) are equipped with AMD Virtex-7 FPGAs. The fourth generation of the UT boards (UT 4) are equipped with either AMD Ultrascale XCVU80, XCVU160, or XCVU190 FPGAs. The FPGAs receive their main clock signal through the Belle II Trigger Timing (B2TT) System [41, 42]. The global clock signal is generated from SuperKEKB's radio-frequency clock of $10.385\,\mathrm{MHz} \cdot 49 = 508\,\mathrm{MHz}$. The divided-by-four clock signal of 127.22 MHz is distributed to all modules of the L1 Trigger. All modules of the L1 Trigger are synchronised to the beam revolution cycle of about 100 kHz, via the *revosig* signal [43]. While phase alignment of all modules would be feasible using this signal, the necessary calibration is not performed in practice.

Configuration, monitoring and maintenance of the L1 Trigger is performed through the Trigger Slow Control [44], an adapted version of the DAQ Slow Control [45]. For this purpose, the main FPGA on the UTs is connected to a host CPU via the Versa Module Eurocard (VME) protocol [46]. Communication between host and FPGA is established through kernel-level Linux drivers provided by the Belle II Trigger Group. The host is connected via Ethernet to the internal Belle II network, where an interface is exposed via the Network Shared Memory 2 (NSM 2), successor to its first version [47].

2.1.2 Central Drift Chamber

The CDC is the primary device for tracking and charged particle identification at the Belle II Experiment. Its main component is a cylindrical drift chamber filled with a 50% helium, 50% ethane mixture and immersed in a magnetic field of 1.5 T. At the end of the chamber in the negative z-direction sits the backward endcap, and in the positive z-direction sits the forward endcap. Inside the chamber, wires are drawn along the z-axis between these endcaps. Two types of wires are used: field wires made of aluminium with a diameter of 126 µm and sense wires made of gold-plated tungsten with a diameter of 30 µm. In total, there are 42 240 field wires and 14 336 sense wires. [31]

The configuration of the sense wires is shown in table 2.2. Sense wires are arranged in 56 layers and eight superlayers to differentiate between axial and stereo layers. Axial layers are perpendicular to the detector's z-axis. Stereo layers are tilted at an angle relative to the beam axis, enabling estimation of the z-component of a reconstructed track's momentum.

Table 2.2: Configuration of the CDC sense wires. Taken from ref. [5].

Super Layer	No. layers	Sense wires per layer	Radius [mm]	Stereo angle [mrad]
Axial 1	8	160	168.0 – 238.0	0.
Stereo 2	6	160	257.0 – 348.0	45.4 – 45.8
Axial 3	6	192	365.2 – 455.7	0.
Stereo 4	6	224	476.9 – 566.9	-55.3 – -64.3
Axial 5	6	256	584.1 – 674.1	0.
Stereo 6	6	288	695.3 – 785.3	63.1 – 70.0
Axial 7	6	320	802.5 – 892.5	0.
Stereo 8	6	352	913.7 – 1003.7	-68.5 – -74.0
Axial 9	6	384	1020.9 – 1111.4	0.

For data readout, the analog sensor signal from each sense wire is amplified and digitised [48]. As soon as the amplitude exceeds a predefined threshold, the timing and amplitude information, along with the time-over-threshold, are extracted as scalar values [49]. The amplitude information is a summation of all sampling points above the threshold. For the DAQ system, up to 25 points are summed up into the amplitude value. For the L1 Trigger, up to 3 points are summed up into the amplitude value.

Based on the fitted waveform, the drift time is extracted as timing information, and the amplitude is extracted as voltage. The digitised amplitude and timing values for each value

are forwarded to the Belle II DAQ System and the CDC L1 Trigger. In the following, I will focus on the L1 Trigger, as it is relevant for this work.

2.1.3 Central Drift Chamber Trigger System

The CDC L1 Trigger reconstructs momentum and position of tracks originating near the interaction point by analysing the input data from the sense wires [50]. The trigger receives a reduced subset of the CDCdetector: For superlayer one, the five outermost layers are sent to the trigger. For all other superlayers, the five innermost superlayers are sent to the trigger. Figure 2.3 shows a system-level overview of the CDC L1 Trigger. The trigger chain consists of six boards. The dataflow runs from left to right. Each board implements a stage in the trigger pipeline.

The Analog Detector Frontends represent the first stage in the CDC trigger system. In total, there are 300 boards, each receiving data from 48 sense wires. Its main components are an ASIC for readout and amplification of the sense wires, an Analog Digital Converter (ADC) with 10 bit resolution and a sampling rate of 31.75 MHz, and a FPGA for signal processing. The dataflow for the CDC L1 Trigger is split between DAQ and L1 Trigger. Digitised amplitude and timing values for the L1 Trigger are forwarded with reduced precision and spatial resolution due to bandwidth limitations on the physical link. It is forwarded to the next stage through GTs via a low-latency custom protocol [50].

The Merger Modules represent the second stage, composed of 70 boards. This stage uses UT 3 boards equipped with FPGAs. Their main purpose is to aggregate sense wire information into spatial chunks, thereby reducing the number of physical connectors in subsequent trigger stages.

The Track Segment Finders receive sense wire information for each complete superlayer. In total, there are nine UT 4 boards for this pipeline stage. It implements a pattern-matching logic, triangular for the innermost layer, hourglass shape for the outer layers, to identify track segments that belong to track candidates [51]. Track segments are the fundamental building block of all subsequent CDC track-trigger algorithms.

The 2D Fitter receives track segments from the Track Segment Finder. Four UT 4 boards each process a half-detector, effectively leveraging spatial parallelism to increase the available system resources at this stage without increasing latency. A two-dimensional Hough transform in the x-y plane is applied to the boards to predict tracks originating from the interaction point. The resulting data is used in the 3D Fitter and the GDL.

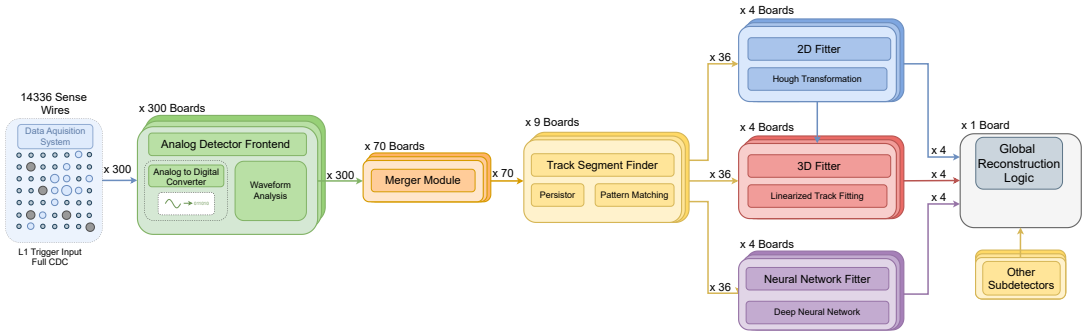


Figure 2.3: System-level block diagram of the CDC L1 Trigger at Belle II.

The 3D Fitter receives track segments from the Track Segment Finder and preprocessed information from the 2D Fitter. Four UT 4 boards enable spatial parallelism in the same manner as for the 2D Fitter. The 3D Fitter additionally predicts the longitudinal impact parameter for the tracks by incorporating wire information from the axial and stereo layers of the CDC [52]. Its results are forwarded to the GRL.

The Neural Network Fitter is the most recent addition to the CDC L1 Trigger, receiving track segments from the Track Segment Finder. Again, its implementation is distributed on four UT 4 boards. Track segments from all superlayers are used as input to a Deep Neural Network (DNN) to predict the z-offset and polar emission angle, and to train a classifier for each track in a trigger timing window [53, Bäh25]. The predicted results are forwarded to the GRL.

Although not yet validated in the Belle II Experiment, conceptual studies are ongoing to support track finding using displaced vertices in the CDC L1 Trigger. The current trigger systems reject tracks that do not extrapolate to the interaction point or do not hit enough superlayers, thereby avoiding trigger signals in response to beam background or cosmic rays. In an effort to support displaced vertices, the Track Segment Finder [Ung23b] is reworked to implement a multi-three-dimensional Hough transformation [Ung25]. Additional studies using CNNs for identifying tracks with displaced vertices have also been considered [Ung23a].

2.1.4 Electromagnetic Calorimeter

The ECL detector consists of 8736 CsI(Tl) crystals that produce scintillation light in response to energy deposited by charged particles and electromagnetic showers initiated by photons.

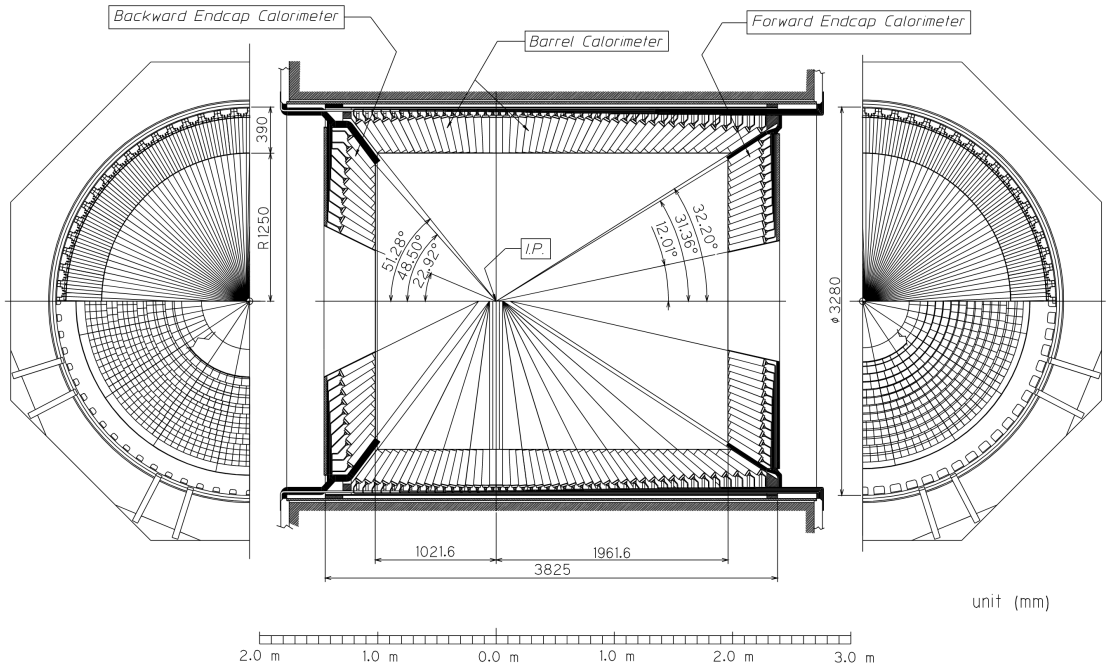


Figure 2.4: Schmeatic of the ECL detector at Belle and Belle II. Taken from ref. [27].

An overview is shown in figure 2.4. The detector is split into three regions, barrel, forward endcap, and backward endcap, containing 6624, 1152, and 960 crystals, respectively. The centre of the figure shows the cross-section at y = 0.0 m in the x-z plane. The left side of the figure shows a cross-section at z = −1.02 m of the backward endcaps in the y-z plane. The right side of the figure shows a cross-section at z = 1.96 m of the forward endcaps in the y-z plane. Crystals are tilted towards the interaction point by an offset of 1.3° in the ϕ and θ directions, thereby preventing photons from escaping between the crystal gaps. Overall, the detector covers a total solid angle of 91 % of 4π. Photons with ultra-violet to visible wavelengths are detected by two p-type, intrinsic, n-type (PIN) photodiodes mounted on the crystal's rear surface. The electrical analog signal is amplified near the diode, minimising electromagnetic interference. [27, 34] For data acquisition during the experiment, all analogue outputs of the crystals are connected to the Belle II DAQ System and the ECL L1 Trigger. In the following, I will focus on the L1 Trigger, as it is relevant for this work.

2.1.5 Electromagnetic Calorimeter Trigger System

The Electromagnetic Calorimeter Trigger System provides cluster information based on energy depositions of charged particles, as well as ECL-specific trigger bits. An overview

of the ECL L1 Trigger is shown in figure 2.5. The trigger chain consists of six boards. The dataflow runs from left to right.

The Shaper-Digital Signal Processor and Collector Module (Shaper-DSP) is a custom board consisting of analog Integrated Circuits (ICs) and a small FPGA for configuration. In total, there are 576 boards in the ECL L1 Trigger. Each board receives the signals from 16 crystals. Each signal is first amplified and then calibrated for light-yield differences between crystals. An analog sum of 16 adjacent crystals, so-called Trigger Cells (TCs), reduces the number of signals from 8376 to 576. Finally, TCs are sent to the next stage. [54]

The Flash ADC Analog Module (FAM) is another custom board consisting of ADCs and a FPGA for signal processing. In total, there are 52 boards in the ECL L1 Trigger. Each board receives the signals from 12 Shaper-DSPs. Signals from the Shaper-DSPs are first digitised via an ADC with 12 bit resolution and a 100 MHz sampling rate, then forwarded to the FPGA for further processing. On the FPGA, the discrete-time series is split into 127.2 ns timing windows, each containing twelve discrete signal points. The separation into timing windows allows for a synchronisation of the readout system from 100 MHz at the ADC to the 127.22 MHz base clock of the L1 Trigger system. A χ^2 waveform fit is applied to each timing window to extract amplitude and timing information [55]. The amplitude is calculated from the waveform peak. It is given as 12 bit fixed-point number with a resolution of 5.25 MeV. TCs below a 100 MeV threshold are masked to zero, as these signals are considered background for the following stages. Timing information is given relative to the timing window clock counter, which is synchronised over all FAMs. It is given as 7 bit fixed-point number with a resolution of 0.97 ns. The extracted amplitude and timing information are forwarded to the next stage. [54]

The Trigger Merger Module (TMM) is based on the UT 3. In total, there are seven boards in the ECL L1 Trigger. Each board receives signals from eight FAMs. Its main purpose is to move data via high-speed GTs from FAMs to the subsequent stage. A direct aggregation of all physical links is not feasible, as the number of FAMs exceeds the number of available transreceivers on commercially available FPGAs.

The ICN-ETM is based on the UT 4. This stage receives all aggregated TCs via seven TMM in the ECL L1 Trigger. The ICN-ETM implements a clustering algorithm for TCs based on the isolated-cluster logic from Belle [56]. Isolated regions in the ECL are identified through pattern matching, where each TC is identified by a binary value. A TC is considered hit when the amplitude threshold of 100 MeV is reached. Based on the hit value of adjacent TCs, up to six cluster candidates per timing window are derived through a Boolean logic network. The

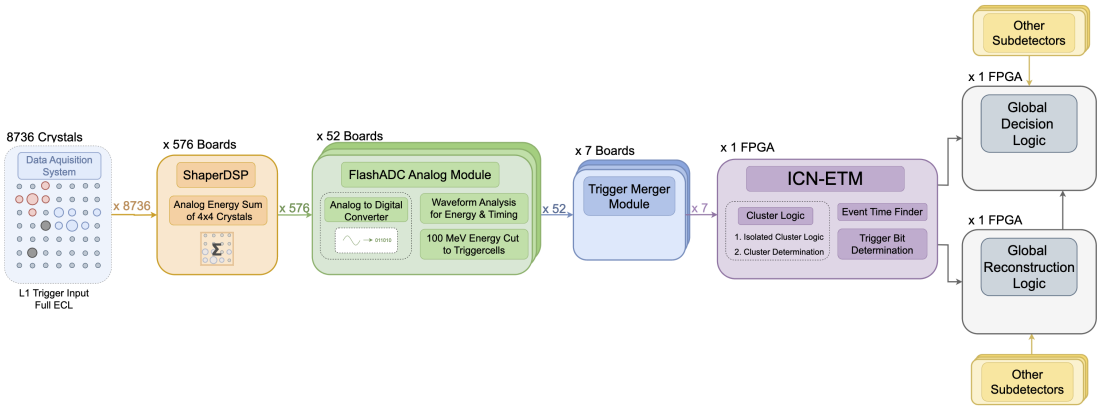


Figure 2.5: System-level block diagram of the ECL L1 Trigger at Belle II. Taken from ref. [1].

limitation to binary inputs enables a hardware-efficient implementation on FPGAs. In total, up to six clusters are calculated per trigger window and used to determine trigger bits subsequently. Trigger bits are sent to the GDL as well as the GRL. In addition, the position, energy and timing information of up to six clusters is sent to the GRL.

The technical specifications for the ICN-ETM are summarised below. Table 2.3 summarizes the technical specifications. The whole digital processing chain in the ECL L1 Trigger, starting with the FAM, is operating on a 127.2 ns timing window, i.e., the full subdetector is processed with a rate of 8.00 MHz. The system is deadtime-free because it processes consecutive trigger windows without delay. In addition, the complete L1 Trigger must adhere to the latency requirement of 5 µs. As this latency includes the data transfer between boards ($\approx$ 400 ns), waveform fitting ($\approx$ 700 ns), and other preprocessing steps, the actual latency available for clustering algorithms in the ECL L1 Trigger is much lower [54]. Because the actual latency budget available to a clustering algorithm in the ECL L1 Trigger is unknown, I measure the current ICN-ETM during operation.

Table 2.3: Technical specification of the ICN-ETM.

Technical Specification	
Throughput	7.86 MHz
Latency	676 ns
Deadtime	0 ns
System Frequency	127.22 MHz
FPGA	AMD XCVU 160
Board	UT 4

2.2 Deep Learning on Point Clouds

Multi-dimensional sensors such as depth-red-green-blue imaging sensors, lidar, radar, or application-specific distributed sensor systems enable the capture of spatial information for a wide range of applications [57]. Data generated by these sensors is generally available as multi-dimensional point clouds. Compared with two-dimensional imaging sensors, higher information density promises improved system performance in terms of accuracy and robustness for applications such as shape classification, object tracking, semantic segmentation, and object classification [58].

A formulation of a point cloud for perception tasks is given in equation (2.1) and will be used throughout the thesis. A point cloud $\mathcal{P}$ is defined as a set of n feature vectors. Each point $p \in \mathcal{P}$ is represented by an F-dimensional feature vector $\vec{p}$. Depending on the application, the number of points $n = |\mathcal{P}|$ may vary. To represent the point cloud in a single data structure, I define a matrix $A \in \mathbb{R}^{n \times f}$. I will use this representation to describe data from distributed sensor systems, where a point $\vec{p}$ represents a sensor with distinct features, such as its spatial coordinates and additional data, such as intensity, colour, or depth. In the following and throughout this thesis, these point clouds serve as input for machine learning tasks.

$$\mathcal{P} = \{\vec{p}_0, ..., \vec{p}_{N-1}\} \subseteq \mathbb{R}^F \tag{2.1}$$

Deep learning methods on point clouds, hereafter referred to as PCNs, have been classified into distinct categories by various literature reviews [58, 59, 60, 61]. I have performed a taxonomy harmonisation to consolidate the varying classification schemes, resulting in the following categories: point-based, projection-based, volumetric-based, and graph-based methods. An overview is shown in figure 2.6.

Point-based PCNs operate directly on the raw point cloud data, thus avoiding transformation to other representations. Three notable PCN architectures are PointNet, PointNet++, and PointMLP. PointNet [62] is an early work that uses this approach to process unordered sets and achieve permutation invariance via symmetric functions [61]. PointNet++ [63] further improves classification performance by introducing a sampling layer that aggregates local geometric features for each point. PointMLP [64] is a pure residual Multi Layer Perceptron (MLP) which achieves better trainability and thus supports deeper network topologies than earlier works.

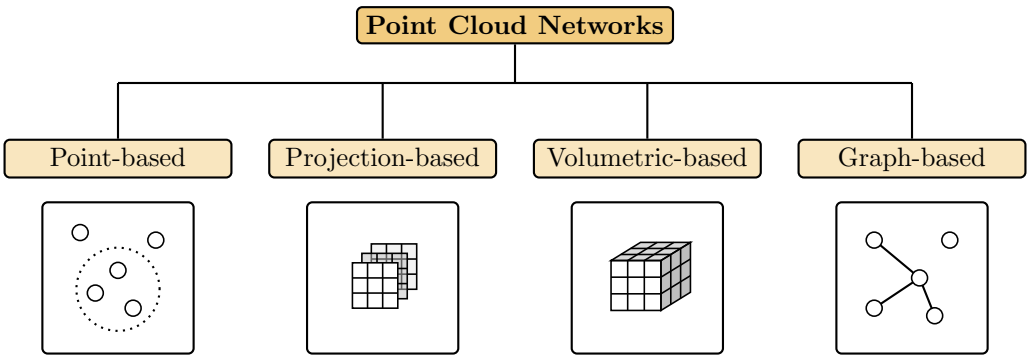


Figure 2.6: Harmonized Taxonomy of Point Cloud Networks.

Projection-based PCNs map the multi-dimensional point cloud into multiple two-dimensional planes. Each plane represents a view into the point cloud from a certain direction. By merging multiple such planes into a single feature matrix, a global representation is determined. Two distinctive works that use this approach are the Multi-View CNN [65] and the Group-View CNN [66]. This approach has shown two advantages. First, the resulting global representation has fixed dimensions across all point clouds, thereby increasing regularity in successive computations. Second, this approach allows the reuse of existing neural networks for two-dimensional image classification and segmentation.

Volumetric-based PCNs extend directly on the idea of two-dimensional CNNs, by mapping a point cloud into a multi-dimensional grid of so-called cubed voxels. Each point is assigned to the closest voxel centre in the 3D grid [60]. Converting a point cloud into a three-dimensional grid of cubed voxels is called voxelisation. Recent works that use voxelisation include MKConv [67] and CCMNET [68].

Graph-based PCNs construct an intermediate graph representation from the point cloud, adapting layer reuse concepts from GNNs. An early example is the Dynamic Graph Convolutional Neural Network [69], which uses graph convolutions to aggregate local and global features. Domain-specific variants, such as ParticleNet [70] for particle physics, have since followed. Given its relevance to large-scale scientific experiments and to this thesis, this approach is detailed below.

Graph-based PCNs adapt the idea of reusing layers from GNN by first building an intermediate graph representation from the point cloud. An early work of this kind is the Dynamic Graph Convolutional Neural Network [69]. Graph convolutions represent sampling layers, aggregating local and global features. Adaptations for application domains, e.g.,

particle physics, have been presented in Particle Net [70]. As this approach is prominent in large-scale scientific experiments, graph-based PCNs are described in more detail below.

2.2.1 Graph-based Point Cloud Networks

Graph-based PCNs, or more generally, GNNs on multi-dimensional point clouds, have been discussed in the literature [71]. Because graph-based PCNs reuse existing layers from general GNNs, overlapping terminology exists. The goal of this section is to provide unified definitions of the layers and topologies that occur in such networks. Generally, applying graph-based learning methods to point clouds is a two-step approach. First, a suitable graph representation of the input point cloud must be generated. In the following, this step will be referred to as graph building. Second, GNN methods can be executed on this generated graph. A generic compute model which has found wide adoption in the field is message passing [72]. For graph building, consider the previously defined point cloud from equation (2.1).

A subset of query points $\mathcal{V} = \{\vec{v}_0, \dots, \vec{v}_{M-1}\} \subseteq \mathcal{P}$ is selected from the point cloud. In the simplest case, $\mathcal{V} = \mathcal{P}$ and each point is a query point. A bipartite graph $\mathcal{G}(\mathcal{V}, \mathcal{P}, \mathcal{E})$ is then constructed, where each directed edge $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{P}$ connects a query point to one of its nearest neighbours in $\mathcal{P}$. The result of the nearest-neighbour algorithm is a local neighbourhood $\mathcal{N}(i) \subseteq \mathcal{P}$ for every query point v_i. Additionally, an edge feature function $\psi : \mathcal{E} \to \mathbb{R}^d$ is defined that maps each edge $(v_i, v_j) \in \mathcal{E}$ to a feature vector $\vec{e}_{j,i} = \psi(\vec{v}_j, \vec{v}_i)$ of dimension d. The definition of local neighbourhoods and the edge feature vectors constitute the result of the graph-building step and are required for the subsequent message-passing step.

Message passing is a generalised formula for describing graph convolutions in irregular domains. It is widely used in compute frameworks for GNNs such as PyTorch Geometric [73]. The message-passing operation is defined locally, meaning the transformation is applied independently at each query point v_i. The local formulation of the message-passing step is shown in equation (2.2).

For a network layer k, the set of query vertices $\mathcal{V}$, the local neighbourhood $\mathcal{N}(i)$, and the edge feature vectors $Vece_{j,i}$ are given. The update function $\gamma(\cdot)$ and the message function $\phi(\cdot)$ are differentiable and learnable. $\bigoplus(\cdot)$ is a differentiable, permutation-invariant function, e.g., sum, mean or max. The output of this transformation is an updated feature vector for each vertex, which is subsequently fed to the next neural network layer.

$$\vec{v}_i^{(k+1)} = \gamma_i^k \left(\vec{v}_i^k, \bigoplus_{j \in \mathcal{N}(i)} \phi^k \left(\vec{v}_i^k, \vec{v}_j^k, \vec{e}_{j,i} \right) \right) \tag{2.2}$$

Using the definitions of the graph-building step and the message-passing step, two topologies of graph-based PCNs can be distinguished based on the arrangement of the layers in the network:

The topology of a static graph-based PCN is shown in figure 2.7. A simplified network with three layers is shown. The dataflow is oriented from left to right. The graph-building step is executed in the first layer and is then reused in all subsequent message-passing layers of the network. In the literature, this approach is also described as a generic GNN with a dedicated preprocessing step that converts the point cloud into a graph representation [21].

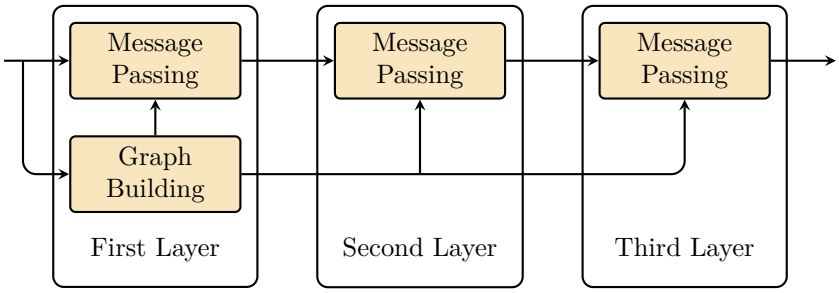


Figure 2.7: Topology of a static graph-based Point Cloud Network.

The topology of a dynamic graph-based PCN is shown in figure 2.8. Again, a simplified network with three layers is shown, and the dataflow runs from left to right. Compared with the static topology, a graph-building step is inserted at every layer of the network. Networks that use this topology include, for example, the previously mentioned Dynamic Graph Convolutional Neural Network [69], the GravNet [74], and the AGConv [75].

Comparing the two network topologies makes it clear that a different approach is required for deploying these networks to ensure efficient execution. In the static topology, depending on the application, all information for the graph-building step may already be available at design time. For the dynamic topology, this optimisation is generally infeasible. Using the terminology defined above, the Interaction Network, the GravNet, and Object Condensation will be described in more detail in the following subsections.

Comparing the two network topologies makes it clear that deploying these networks

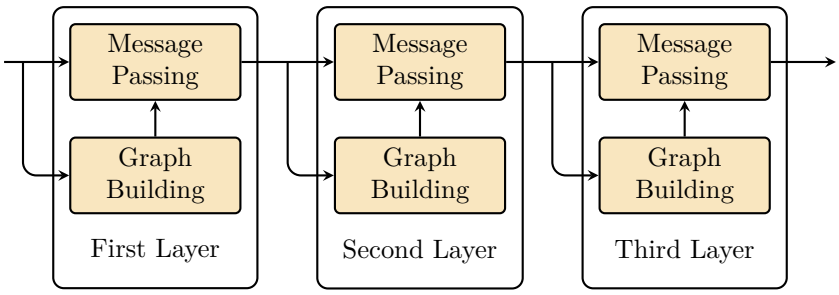


Figure 2.8: Topology of a dynamic graph-based Point Cloud Network.

efficiently requires a tailored approach. For the static topology, depending on the application, all information for the graph-building step may already be available at design time, enabling precomputation of the graph structure. For the dynamic topology, this optimisation is generally unfeasible since the graph must be reconstructed at each layer.

Two examples of these topologies, the Interaction Network for static graph-based PCNs and the GravNet for dynamic graph-based PCNs, will be described in the following subsections as they will be used throughout the thesis. In addition, an overview of the object condensation algorithm is given, which is used for clustering on PCNs.

2.2.2 Interaction Network

The Interaction Network captures local and global relations through graph representations [76]. It has been applied to physical systems such as mass-spring systems and rigid-body systems, as well as to data from irregular image sensors [77]. While the Interaction Network operates on graph data and is thus considered a GNN, it has been applied to point cloud processing, particularly for irregular detector geometries [22]. Notably, Interaction Networks have demonstrated strong performance in particle tracking via edge classification [21]. In this work, I consider the combination of the Interaction Network with a suitable graph-building step to be a static graph-based PCN.

The Interaction Network is concisely described by adapting the message-passing formulation in equation (2.2) from the previous section. Table 2.4 shows the required definitions. Both message and update components are realised by MLPs. For aggregation, a summation operand is used in the original version from ref. [76]. The computational complexity scales linearly with the number of vertices and the number of edges: $\mathcal{O}(|\mathcal{V}|+|\mathcal{E}|)$.

Table 2.4: Local formulation of message passing used in the Interaction Network.

Component	Symbol	Function	Description
Message	ϕ^k	MLP	Relational model
Aggregation	$\bigoplus$	$\sum$	—
Update	γ^k	MLP	Object model

2.2.3 GravNet

The GravNet is a dynamic graph-based PCN for multi-particle reconstruction in Electromagnetic Calorimeter [18, 74]. It should not be confused with the similarly named GarNet, which is derived from the original GravNet model [78]. While GravNet was originally developed for applications in the CMS Detector, it has also been studied for the Electromagnetic Calorimeter in the Belle II Detector [19].

The GravNet layer follows the typical structure of dynamic graph-based PCNs, consisting of a graph-building step followed by a message-passing step. The graph building is performed in a learned latent space representation $S = \{\vec{s}_0, \dots, \vec{s}_{N-1}\}$, derived from the input point cloud $\mathcal{P}$ through the linear function $F_S : \mathcal{P} \to \mathcal{S}$. The neighbourhood for each point is determined through the k-nearest-neighbour algorithm in equation (2.3), where $d(s_j, s_i)$ denotes the Euclidean distance in the latent space. The edge features for the generated graph are computed as $\vec{e}_{j,i} = e^{-10d(\vec{s}_j,\vec{s}_i)}$.

$$\mathcal{N}(i) = \underset{\mathcal{J}\subseteq\mathcal{S},|\mathcal{J}|=k}{\arg\min} \sum_{j\in J} d(\vec{s}_j, \vec{s}_i) \tag{2.3}$$

Using this underlying graph, a message-passing step is performed with the components summarised in table 2.5. The messages exchanged in the local neighbourhood are linearly transformed into a feature space and multiplied by the weighted distance from the query vertex. For aggregation, both summed and max-pooled features are extracted and concatenated. The vertex update is then performed through a final linear layer.

Table 2.5: Local formulation of message passing used in GravNet.

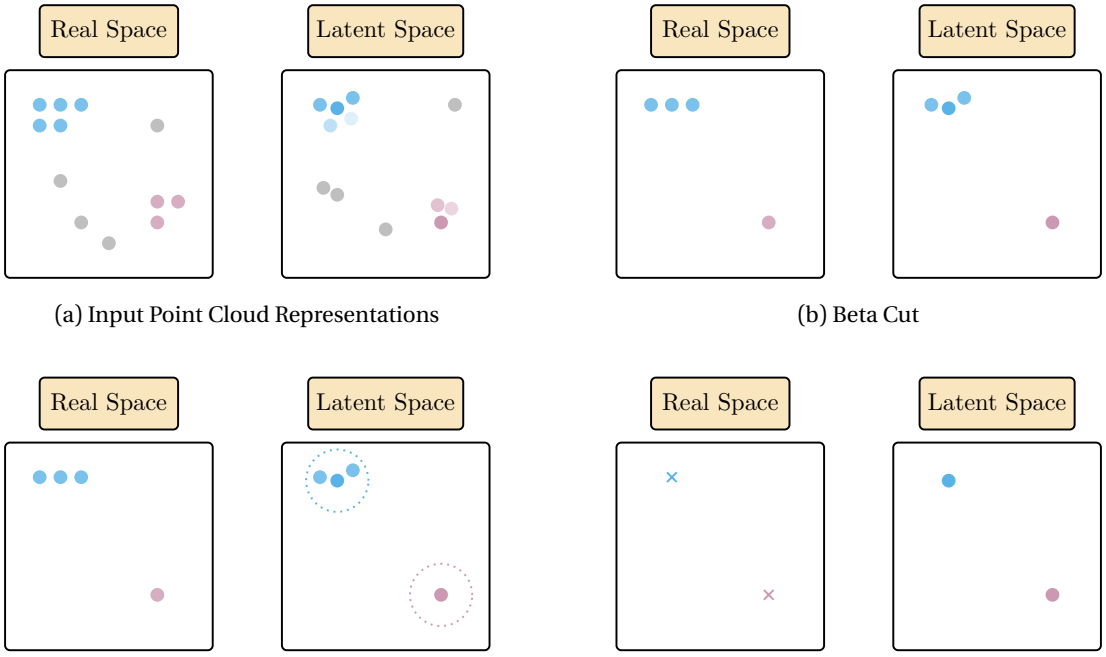
Component	Symbol	Function	Description
Message	ϕ^k	$\vec{e}_{j,i} \cdot \text{MLP}(\vec{v}_j^k)$	Distance weighting of feature space
Aggregation	$\bigoplus$	$\max \vert\vert \sum$	Concatenation of sum and max-pooled features
Update	γ^k	MLP	Linear transformation into output representation

The k-nearest-neighbour algorithm dominates the computational complexity of GravNet. In a naive implementation, the distances between all pairs of vertices are calculated, resulting in an overall complexity of $\mathcal{O}(|\mathcal{V}|^2)$. While implementations with a worst-case complexity as low as $\mathcal{O}(k|\mathcal{V}|\log|\mathcal{V}|)$ exist, their parallelisation is limited [79].

2.2.4 Object Condensation

The object condensation algorithm is a one-shot clustering and detection algorithm on point clouds in high-energy physics [80]. In this thesis, it is used in electromagnetic calorimeters for identifying and clustering energy depositions. The clustering algorithm allows for the clustering of an a priori unknown number of energy depositions in the detector, without requiring the number of clusters to be specified. Compared with previous clustering approaches such as DBSCAN [81], it is better suited to varying conditions in the dataset because it operates on a learned feature space. In addition, the object condensation algorithm is fully deterministic, unlike DBSCAN, since its result does not depend on the ordering of points in the input point cloud.

Figure 2.9 shows the inference of the object condensation algorithm in two steps. As input, the algorithm requires both a real-space and a latent-space representation of a point cloud (see section 2.2.3). Each point is associated with a beta value $\beta \in [0,1]$. The beta value is predicted by the previous network layers and is trained such that one representative point of each cluster is assigned a high beta value [1]. An example visualisation is shown in figure 2.9a. In the first step in figure 2.9b, a beta cut is applied through a simple threshold function. All points above this threshold are subsequently considered candidates for condensation points. The beta threshold is a runtime parameter of the inference algorithm. In the second step in figure 2.9c, condensation points are selected from the candidates by calculating the distance in latent space to all other condensation point candidates. An isolation threshold is applied, and a candidate remains a condensation point only if there are no other candidates within the isolation radius with a higher beta value. The isolation threshold is a runtime parameter of the inference algorithm. The remaining condensation points each represent a cluster and contain the predicted cluster-level features. Figure 2.9d shows the clusters as crosses in the real space. In this example, exactly two clusters are identified by the algorithm.



(a) Input Point Cloud Representations

(b) Beta Cut

(c) Condensation Point Candidate Selection

(d) Resulting Cluster Representation

Figure 2.9: Step-by-step overview of the inference step of the object condensation algorithm. Real and latent spaces are two-dimensional for visualisation purposes. The input point cloud contains points belonging to two clusters. Clusters are shown in blue and red, respectively. Points not belonging to any cluster are shown in grey. Adapted from Refs. [1, 17].

2.3 Deployment Targets

This chapter gives an overview of suitable deployment platforms for machine learning workloads in hard real-time, high-throughput applications such as hardware trigger systems. Section 2.3.1 introduces the island-style FPGA architecture, whereas section 2.3.2 describes tile-based CGRA architectures. Section 2.3.3 explains how FPGA and CGRA partitions can be integrated into a single heterogeneous SoC, alongside CPU cores and other peripherals.

2.3.1 Field Programmable Gate Arrays

An Field Programmable Gate Array (FPGA) is a programmable logic device, a type of IC that can implement any digital circuit [82]. In contrast to ASICs, an FPGA can be reconfigured on the fly without requiring dedicated hardware to be manufactured. To achieve fine-grained reprogrammability, FPGAs contain a large number of functional elements connected via routing networks [83]. Figure 2.10 shows the architecture of an island-style FPGA. The figure is simplified to avoid visual clutter and highlight the functional elements inside the IC. The global routing network consists of switch boxes and interconnects with numerous parallel channels, which allow functional elements in the FPGA to be connected freely. In our exemplary architecture, I consider five different types of functional units: Logic clusters, input output (IO) elements, interposer elements, Random-Access Memory (RAM) elements, and Digital Signal Processing (DSP) elements.

Logic clusters contain registers, e.g., flip-flops, as well as basic logic elements. Nowadays, logic elements are typically implemented as Lookup Tables (LUTs) to realise any Boolean equation. Typical sizes for LUTs are between four and six inputs per output, as this minimises the logic delay and the area of the logic table [82].

IO elements connect to external pins on one side and to internal wiring on the other. Depending on their purpose, the drive strength varies, or additional control logic is supplied. As an example, tri-state buffers are typically required for interfacing external bus systems.

Interposer elements are used when an FPGA consists of multiple dies that are integrated into a larger platform. Interposers are a bottleneck in routing and introduce an additional delay of around 1 ns [85]. Thus, their availability must be considered when implementing efficiently on large FPGAs.

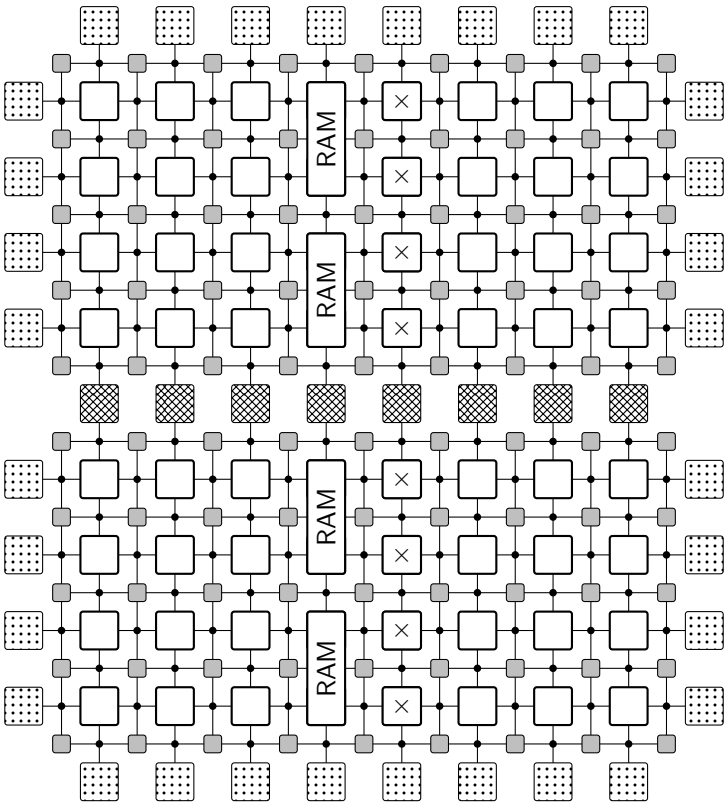


Figure 2.10: Overview of an island-style FPGA, consisting of two chiplets connected via Silicon Stacked Interposer (SSI). The routing network is composed of connection lines, interconnects as dots, and switch boxes in grey. Logic clusters are white, IO elements are dotted, interposer elements are crosshatched. Special functional units depicted are RAM and DSP elements. Visualisation adapted from ref. [84].

RAM elements are special elements on the FPGA, able to store large amounts of data. They are composed of dedicated large static RAM modules. Storing data in RAM elements is more area-efficient than in distributed logic clusters, at the cost of limited read and write ports for accessing the data.

Digital Signal Processing elements are dedicated elements to implement certain mathematical operators more efficiently than on distributed logic elements. For example, fixed-point or floating-point multiplication can be efficiently mapped onto these elements, enabling higher system frequency and better area utilisation on the FPGA.

Beyond these configurable functional units, modern FPGAs often feature various additional hard Intellectual Property (IP) cores. These cores are specifically designed to perform a fixed function and cannot be reprogrammed freely. Examples include GTs, media access layer IP cores for high-bandwidth Ethernet, or even full processor subsystems. The inclusion of such

hard IP cores reflects a broader trend of increasing FPGA complexity and chip area to meet the growing demand for compute.

However, it is well known that as chip size increases, wafer yield diminishes [86]. Therefore, semiconductor companies investigate strategies to improve their yield while simultaneously increasing the available chip area. Using the SSI technology, integrating multiple chiplets into a single FPGA is feasible in commercial chips. Pioneering work in this direction has been performed by AMD (formerly Xilinx), enabling multi-chiplet FPGAs since the 2010s [87, 88].

Figure 2.11 depicts three recent architectures from AMD, of which two use the SSI technology to connect multiple FPGA chiplets. In the following, FPGA chiplets on a single chip will be referred to as Super Logic Regions (SLRs). The AMD XCVC1902 is a first-generation Versal SoC used on the AMD VCK190 evaluation board [89]. While it only has one SLR, other components are connected via SSI. The AMD XCVU190 is a device from the Ultrascale generation that is currently used in the Belle II Experiment [90]. It features three vertically stacked SLRs. The AMD XCV80 is a second-generation Versal SoC used on the AMD Alveo V80 [89]. It also features three SLRs in a vertically stacked structure.

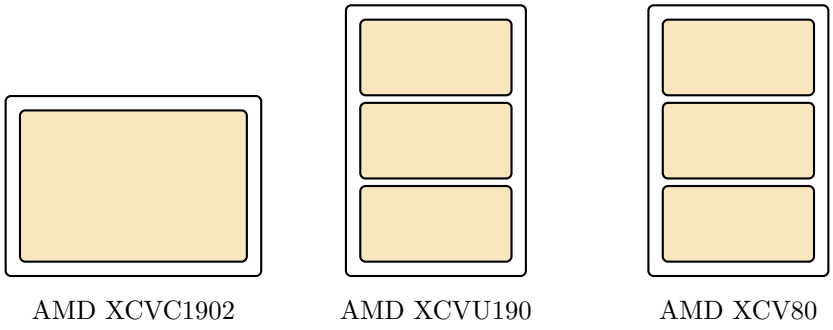


Figure 2.11: Selected commercially available FPGA architectures from AMD with different numbers of SLRs [89, 90].

2.3.2 Coarse Grained Reconfigurable Arrays

The fine-grained reconfigurability of FPGAs comes at a cost, namely the routing fabric required to enable the synthesis of digital circuits. The fraction of the normalised routing delay that is incurred by the routing fabric increases steadily as the technology node size decreases. For 7 nm technology node, the metal delay alone amounts to 80 % of the total delay [91]. In addition, studies have shown that designs on FPGAs are, compared to full-custom ASICs,

approximately 35 times larger and between 3.4 and 4.6 times slower [92]. To overcome these limitations, Coarse Grain Reconfigurable Arrays (CGRAs) employ arrays of coarser tiles that operate on multi-bit data words rather than individual logic cells, thereby replacing bit-level with word-level reconfigurability and reducing configuration overhead and interconnect complexity [93].

Figure 2.12 shows an exemplary CGRA architecture. It is a generic architecture with grid-based tile-to-tile communication (North, East, South, West) and a Network-on-Chip (NoC) [94]. It is composed of three types of tiles: compute, router, and IO.

Compute tiles contain processors and local memory, often used as cache or scratchpad memory. Depending on the CGRA architecture, compute tiles can vary in complexity from simple Algorithmic Logic Units (ALUs) up to full-featured Very Large Instruction Word (VLIW) processors.

Router tiles are dedicated tiles that allow the compute tiles to communicate with other tiles in the grid. They are part of the NoC that orchestrates the compute array.

IO tiles expose direct access from external components to the memory of adjacent compute tiles. Depending on the design, external Double Data Rate Random Access Memory (DDR-RAM) may be accessed by compute tiles via Direct Memory Transfer (DMA) or similar protocols. When the CGRA is directly coupled to an FPGA, ping-pong buffers or streaming interfaces can be used to connect both. One example of such an architecture is the AI Engine fabric on the AMD Versal platform [91].

It should be noted that CGRA architectures vary considerably in practice. While the architecture shown in figure 2.12 uses a NoC and direct neighbours, other designs employ shared register files, or even hierarchical interconnect topologies [95]. Similarly, reconfiguration strategies range from fully static execution, as in systolic arrays, to dynamically reconfigurable designs in which the configuration changes every cycle [93].

Based on recent literature reviews, the Xputer [96] is considered one of the first academic CGRA architectures [93]. In recent years, CGRAs have experienced renewed interest, as they have proven suitable for dataflow-intensive workloads such as the hardware acceleration of DNNs [97].

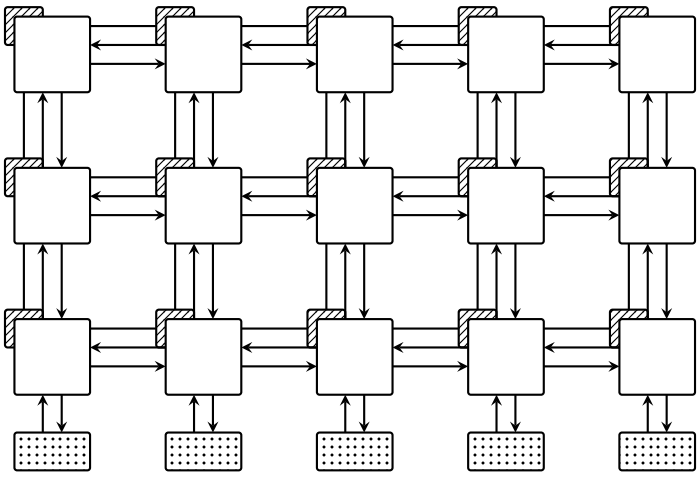


Figure 2.12: Exemplary architecture of a CGRA, adapted from ref. [94]. Compute tiles are white, router tiles are dashed, and IO tiles are dotted.

2.3.3 Heterogeneous System-on-Chips

State-of-the-art platforms for domain-specific deployment, such as machine learning hardware accelerators, combine multiple computer architectures on a single IC [98]. In the following, I refer to such an IC as a heterogeneous SoC.

Figure 2.13 depicts a system-level overview of a heterogeneous SoC, composed of a general-purpose CPU, FPGA fabric, a CGRA hardware accelerator, as well as memory and general purpose input output (GPIO) interfaces. The components of a heterogeneous SoC are connected by a central interconnect, typically a NoC, but depending on the realisation, may also feature one or more bus systems.

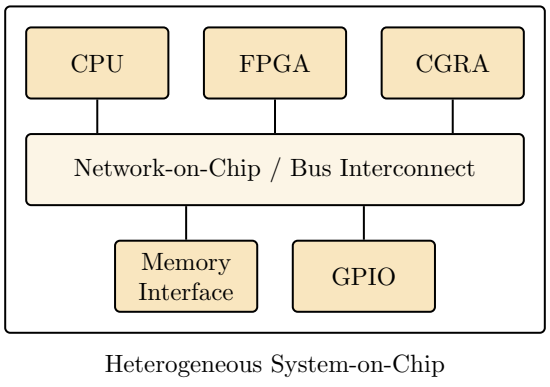


Figure 2.13: Overview of a heterogeneous SoC, based on the definitions used in this thesis.

Heterogeneous SoCs typically achieve better performance and energy efficiency, as operations can be offloaded to the optimal compute architecture for a given workload [99]. For example, control-flow-driven workloads such as system orchestration and scheduling are executed by the CPU. In contrast, dataflow-driven operations such as general matrix-matrix multiplications can be efficiently executed on dedicated CGRA accelerators. If custom operations are required, such as fast Fourier transforms, these workloads can be offloaded onto the FPGA fabric once a suitable compute kernel is designed. Nevertheless, the improved efficiency and performance come at a cost. First, rather than targeting a single compute architecture, it is necessary to implement a dedicated toolflow backend to deploy algorithms on each processor type in the platform. Second, communication and synchronisation between processors incur additional overhead. Especially for low-latency, high-throughput applications, these boundary effects can lead to significant performance degradation. Third, mapping algorithms onto heterogeneous SoCs is a partitioning problem and thus NP-hard [100]. Finding an optimal mapping is computationally expensive, and for large problem instances, the state of the art relies on heuristics to obtain acceptable solutions.

To illustrate these concepts on a concrete platform, the AMD Versal SoC [89, 91] is considered, a commercially available heterogeneous SoC architecture. Figure 2.14 depicts the floorplan of the AMD Versal architecture. Its main parts consist of a Processor and Platform Management Unit (CPU), Fabric (FPGA), and the AI Engines (AIEs) (CGRA). The two CPUs are general-purpose architectures, namely an ARM Cortex A72 [101] and an ARM Cortex R5F [102]. The FPGA fabric provides direct access to hardened features, such as media-access layers for Ethernet and GTs. The AIEs consist of a two-dimensional grid of VLIW processors connected via point-to-point streaming interfaces and a NoC (compare section 2.3.2). As with previous FPGA-only architectures from AMD, the full architecture is built from multiple chiplets to improve yield [85, 86].

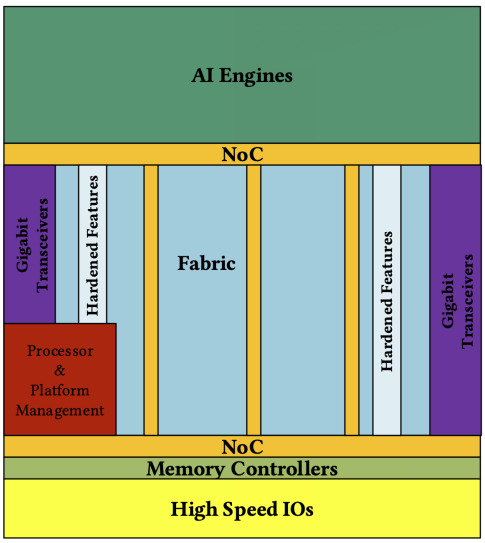


Figure 2.14: Floorplan of the AMD Versal as an example of a commercially available heterogeneous SoC. Taken from ref. [91].

This page intentionally left blank

Chapter 3

Related Work

This chapter surveys the work related to this thesis, structured along two areas as shown in figure 3.1. The first covers application-specific accelerators for static and dynamic PCNs in section 3.1. As both workload classes share irregular data structures, they pose similar acceleration challenges. Focus is placed on FPGAs, CGRAs, and ASICs, as these platforms are best suited for deployment in large-scale scientific experiments. The second covers hardware compilation in section 3.2, distinguishing between template-based and fine-grained approaches. Together, these areas motivate the research gap in current deployment methodologies for point cloud networks, with a synthesis of findings given in section 3.3. Background on PCNs is provided in section 2.2.

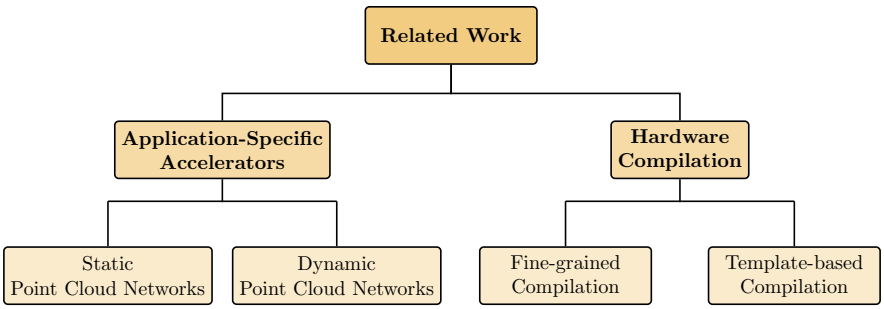


Figure 3.1: Overview of the related work landscape. Application-specific accelerators and hardware compilation form the two pillars motivating this thesis.

3.1 Application-Specific Accelerators

Application-specific hardware accelerators for PCNs have been explored on both ASIC and FPGA platforms. Because the naming of PCNs is sometimes ambiguous and there is no clear definition in the scientific community, the definitions from section 2.2 are used throughout

this section. Thus, some works in the following section refer to GNN accelerators, but they also apply to PCNs.

Static Point Cloud Network. One of the earliest works enabling the acceleration of static PCNs on ASICs is GRIP [103], which adapts a message-passing programming model with user-defined functions to accelerate the convolution step in GNNs. Its architecture is also applicable to static PCNs, where the graph topology is computed once and remains fixed throughout inference. PIMGCN [104] takes a different approach by exploiting ReRAM-based processing-in-memory to overcome the memory bandwidth limitations caused by the irregular access patterns that arise in graph-based sampling, a challenge that is equally present in static PCNs.

Several FPGA accelerators have been proposed for static PCNs, aimed at accelerating comparatively large input datasets. BoostGCN [105] introduces partition-centric feature aggregation with a heavy reliance on off-chip DDR-RAM memory, requiring knowledge of the graph structure at design time and thereby limiting it to static topologies. Liang et al. [106] focus on large sparse graphs in data centre settings, optimising for energy efficiency at a scale beyond the scope of this work, since the application domain differs significantly from that of large scientific experiments. StreamGCN [107] exploits deep pipelining in its dataflow architecture. However, it supports only a subset of static PCN architectures and does not reach the throughput demands of large-scale scientific experiments. Notably, H-GCN [108] is among the first works to leverage AIE on the AMD Versal heterogeneous SoC, partitioning the input graph into dense and sparse regions and offloading the computation to the respective compute units. Compared with a reference implementation on GPU, the H-GCN accelerator achieves lower latency and better energy efficiency. Because the previously mentioned accelerators are designed around application benchmarks from high-performance computing, they generally do not meet the latency and throughput requirements of large-scale scientific experiments. Consequently, the experimental science community has driven a distinct line of research on FPGA accelerators.

Early feasibility studies demonstrated accelerated charged-particle tracking using static PCNs on an FPGA for small point cloud datasets [109, 110]. The Interaction Network (see also section 2.2.2) was deployed as a dataflow accelerator using High Level Synthesis (HLS), and subsequently refined in ref. [111] for the CMS detector at the LHC. The authors report a throughput of 3 million detector snapshots per second, with events consisting of up to 739 points. This is in the same order of magnitude as the requirements of the Belle II hardware trigger. Nevertheless, this accelerator architecture is limited to the Interaction Network topology and does not generalise to other PCN architectures. Further works have addressed

specific applications such as particle identification [112] and jet classification [113], both leveraging the JEDI neural network, a two-layer static PCN. While these designs meet the sub-microsecond latency requirements[1] of the CMS hardware trigger, they support point clouds with up to 50 points only. A subsequent study increases the point cloud size up to 128 while reducing the latency to less than 200 ns [114].

In summary, while previous relevant work on accelerating static PCN exists, this work targets isolated case studies on small datasets. Their deployment has not been validated in large-scale experiments, and their approach is strictly coupled to the underlying data readout system.

Dynamic Point Cloud Network. The acceleration of dynamic PCNs on ASIC has been studied in PointAcc [115] and Point-X [116]. Both works argue that a large fraction of the overall compute time is spent on graph building, sampling, and feature aggregation steps. Thus, they extend common deep learning accelerators based on systolic arrays with custom processing units for these stages. Compared with reference implementations on GPU, both works achieve higher throughput and energy efficiency. Another hardware accelerator designed for dynamic GNNs has been recently presented [117]. While not specifically targeting PCNs such as GravNet, this accelerator supports computing time-variant graphs. Nevertheless, this work targets neither the real-time constraints nor the input scales characteristic of hardware triggers in large-scale scientific experiments.

In addition to these ASIC-based accelerators, FPGA-based accelerators have been studied for dynamic PCNs. DeepBurning-GL [118] is an application-specific hardware accelerator architecture evaluated on the DGCNN [69], a dynamic PCN. While the authors report a performance increase over the reference implementation on CPU, they do not provide absolute inference latencies, precluding a quantitative comparison. Golzar et al. propose a CPU-FPGA architecture supporting inference for the full DGCNN pipeline [119]. However, the presented concept is bandwidth-limited and thus does not scale to the throughput requirements of hardware triggers in large-scale scientific experiments. DGNN-Booster [120] aims to accelerate dynamic GNNs, but implements the required graph construction step as an offline preprocessing step, rendering it unsuitable for real-time applications. A recent work that takes the complete end-to-end processing pipeline into account is Hg-PCN [121], which implements a heterogeneous CPU-FPGA architecture for automotive and human action recognition applications. Graph building, sampling, and sorting are offloaded onto the FPGA partition of the system. Nevertheless, the performance targets

[1] The CMS hardware trigger, similar to Belle II, has a latency deadline in the order of microseconds, but the actual latency for a specific module in the trigger pipeline is typically much lower.

and input scales of this work differ significantly from those required in large-scale scientific experiments.

In the domain of large-scale scientific experiments, Iiyama et al. [78] propose GarNet, an architecture derived from GravNet, in which the dynamic convolution is replaced by a learned weighted aggregation, yielding an MLP-based implementation. However, the graph building is replaced by a learned weighted aggregation, such that no explicit graph building nor message passing is performed at inference time.

In summary, no existing work addresses the complete end-to-end acceleration of PCNs under the latency and throughput constraints imposed by hardware triggers in large-scale scientific experiments.

3.2 Hardware Compilers

The compilation of algorithms, for example, written in C or C++, into synthesizable hardware descriptions has been an active area of research for several decades. HLS toolchains automate this process by scheduling, allocating, and binding resources to generate synthesisable hardware from a high-level algorithmic description. A number of commercial and open-source HLS toolchains have emerged, such as AMD Vitis HLS [122], LegUp [123], and Mentor Catapult [124]. These tools generally rely on static scheduling, generating finite state machines to handle control flow, and therefore require the complete schedule to be known at compile time. An exception is Dynamatic [125], which supports dynamically scheduled circuits through dynamic load and store queues. While this concept is promising, initial studies have indicated limited performance for PCNs [Fre25].

In addition to FPGAs, CGRAs have emerged as an alternative reconfigurable target platform. The compilation process for CGRAs consists of two stages. In the first stage, functions are compiled into binaries which can be executed on compute tiles (see also section 2.3.2). In the second stage, binaries must be mapped onto the grid-like architecture for parallel execution, accounting for data movement and synchronisation between tiles.

In the following, works on hardware compilation are organised according to the taxonomy introduced in section 3.1: template-based approaches, which use predefined architectural templates, and fine-grained approaches, which decompose the application into atomic operations and map them to the target platform.

Template-based Compilation Template-based compilation aims to reuse well-optimised deployment solutions for small subsets of an algorithm, for example, dense layers, to facilitate the deployment of larger algorithms or neural network models. The cost of this approach lies in the lower flexibility for previously unknown applications.

Several template-based frameworks have been proposed for deploying machine learning workloads onto reconfigurable platforms, targeting both FPGAs and CGRAs. FINN is a framework for deploying machine learning workloads on FPGAs, developed and maintained by the AMD research group [126]. It parses ONNX models and generates HLS C++ code targeting AMD Vitis HLS. Only AMD FPGA targets are currently supported. The framework has been steadily extended, for example, for resource-aware machine learning applications in FINN-R [127] and for attention-based neural networks in FINN-T [128]. As of today, it does not support GNNs or PCNs.

hls4ml is a template-based deployment framework specifically developed for trigger applications at CERN [129, 130]. It enables the parsing of ONNX, Keras, and PyTorch models, generating HLS C++ code for common machine learning operators such as dense layers and CNNs. While it enables very low-latency implementations when all operators are supported, it does not support graph-structured inputs or dynamic dataflows. It therefore cannot be applied to GNNs or PCNs.

GNNBuilder [131] and FlowGNN [132] are template-based frameworks specifically targeting GNN acceleration on FPGAs, operating by pattern-matching operators in the compute graph to generate synthesisable hardware. GNNBuilder simplifies the development of GNN accelerators by exposing user-defined message-passing functions and supporting a wide range of GNN architectures. FlowGNN similarly targets GNNs on precomputed graphs, reducing execution latency by enabling parallelism at multiple levels. However, neither framework covers the graph-building or sampling stages, and both execute the model layer by layer, limiting the efficiency of end-to-end pipelines.

Regarding template-based hardware compilation for CGRAs, no architecture-agnostic framework exists in the literature. Available toolchains are tightly coupled to the underlying platform. The thesis, therefore, cover predominantly compilation approaches for the AMD Versal heterogeneous SoC, for which several frameworks have been developed targeting the AIE partition, as well as one architecture-agnostic approach.

A line of work has focused on optimising general matrix multiplications on the AMD Versal AIEs. MaxEVA [133, 134] introduces a template-based approach that partitions large matrix

multiplications into tiled operations, develops separate kernels for elementwise multiplication and adder trees, and uses manual placement to reduce pipeline stalls. AMA [135] builds on the MaxEVA kernels as a baseline, improving throughput and memory bandwidth by explicitly considering the interface between the AIE and FPGA partitions. GAMMA [136] extends this line of work to the second generation of AMD Versal AIEs, with particular attention to buffer placement for minimising memory stalls.

aie4ml [137] introduces higher-level machine learning operators, enabling the deployment of MLPs on the AIE partition with low latency and high throughput. Developed as an extension to hls4ml, it offloads compute-intensive matrix multiplications to the CGRA fabric while retaining the hls4ml frontend for operator parsing. However, the supported operator set is limited to dense machine learning layers and does not extend to graph-structured operators required for PCNs.

A different approach is taken by cgra4ml [138], an architecture design framework for hardware accelerators with a CPU and CGRA partition. The toolchain generates binaries by pattern-matching operations on an input ONNX graph, offloading compute-intensive tasks to a systolic array architecture. As cgra4ml defines and generates the underlying ASIC architecture itself. Designs must be prototyped on a regular FPGA, reducing the practical advantage over a dedicated CGRA deployment. Furthermore, cgra4ml does not provide specific support for the irregular operators occurring in PCNs.

Fine-grained Compilation Fine-grained hardware compilation approaches aim to decompose applications into atomic operations and map them onto the target platform, offering greater flexibility than template-based methods at the cost of increased compilation complexity.

Building on HLS as a backend, ScaleHLS [139] is the first Multi-Level Intermediate Representation (MLIR)-based compiler frontend interfacing with a downstream commercial HLS tool. It implements a transformation from PyTorch to HLS C++, applying optimisation passes to generate pragmas that improve latency, throughput, and compile time. HIDA [140] extends ScaleHLS with additional optimisation passes targeting dataflow structures and is evaluated on ResNet and large language models. However, it supports only a limited subset of data types and offers no mixed-precision support, limiting its applicability in practice. BraggHLS [141] is an MLIR-based compilation flow targeting low-latency networks for experimental science. It employs a custom HLS scheduling algorithm but supports only complete loop unrolling, thereby restricting its applicability to very small networks. None of these frameworks supports graph-structured operators or PCNs.

For CGRAs, fine-grained compilation requires handling both single-tile compilation and the mapping of operations across the grid-like architecture. ML-CGRA [142] presents an MLIR-based compilation framework for fine-grained machine learning acceleration on CGRAs, integrating with existing cycle-accurate simulators such as CGRA-ME for performance evaluation. However, the framework has not been evaluated on real hardware, limiting the practical relevance of its reported results. The foundational infrastructure for the following AMD Versal-specific frameworks is provided by the MLIR-AIE dialect and its associated lowering passes [143]. ARIES [144] is a compiler framework targeting AMD Versal AIEs. It provides a custom placement algorithm to reduce development time and minimise memory stalls, and has been evaluated on MLPs and CNNs. mlir-air [145] is a bare-metal compiler stack for AIEs developed by the AMD research team based on MLIR. It provides low-level access to the configuration registers of the AMD AIEs fabric, thereby enabling a full custom implementation flow. However, it does not provide a complete deployment flow from a high-level model representation to hardware, and is therefore primarily intended as a low-level infrastructure component.

None of the fine-grained CGRA compilation frameworks discussed above has been evaluated on graph-structured or irregular workloads such as PCNs.

3.3 Comparative Analysis

The works surveyed in section 3.1 and section 3.2 are compared in table 3.1 along three overarching axes: general platform and methodology properties, support for PCNs, and evaluation scope. For the general properties, the target platform, methodology, and approach are considered. Dataflow architecture is included as a dedicated criterion because deep-pipelined dataflow execution is essential to meet the latency requirements of hardware triggers in large-scale scientific experiments. Regarding PCN support, static and dynamic PCNs are considered separately. Note that support for dynamic PCNs implicitly includes support for static PCNs to some extent, as the underlying operators are a subset of those required for dynamic inference. For the evaluation, a distinction is made between works evaluated on individual network layers and those demonstrated on complete networks, as the associated complexity differs significantly. Furthermore, end-to-end integration is considered, distinguishing works validated in an operational environment from those evaluated in isolation.

Based on the reviewed literature, the following research gaps are identified. ASIC-based

designs are impractical for hardware triggers in large-scale scientific experiments due to limited market size and long design cycles. The scientific community therefore relies on FPGA-based platforms, for which deployment frameworks such as hls4ml have been validated in operational experiments [20, Bäh25]. However, these frameworks do not support the irregular operators required by PCNs, such as graph building and neighbourhood aggregation. Template-based compilation flows targeting GNN operators exist in GNNBuilder [131] and FlowGNN [132], but neither covers online graph building nor satisfies hardware trigger constraints. Dynamic PCNs have been proposed for hardware-trigger algorithms, such as clustering and track reconstruction, yet no deployment has been reported in large-scale scientific experiments. No compilation methodology for deploying PCNs on CGRAs architectures such as AIE has been proposed. The only end-to-end demonstration on such hardware targets automotive applications [121] and does not constitute a reusable deployment flow.

Taken together, no existing work addresses the deployment of static and dynamic PCNs under the latency and throughput constraints imposed by hardware triggers, using a compiler-based methodology applicable across network architectures. This thesis addresses this gap along four directions. First, a template-based compilation flow for PCNs targeting large-scale scientific experiments is developed. The prevalence of application-specific accelerators in the surveyed literature indicates that predictable performance close to the theoretically achievable performance is required. A template-based approach provides a practical tradeoff between generality and the low-latency, high-throughput application requirements. Second, end-to-end integration is achieved by providing deployment methodologies that encompass not only the neural network itself but also the associated pre- and post-processing stages. Third, the feasibility of the proposed approach is demonstrated in an actual operating experiment. Fourth, compilation support for heterogeneous SoC platforms that combine FPGAs and CGRAs, such as AIE, remains limited, particularly for high-performance applications. This thesis addresses this by developing a deployment methodology for PCNs on such architectures.

Table 3.1: Comparison of related work on application-specific accelerators and compiler-based deployment flows for PCNs in large-scale scientific experiments. ✓ = supported, (✓) = partial supported, ✗ = not supported

Work	General				PCN Support		Evaluation		
	Platform	Methodology	Approach	Dataflow	Static	Dynamic	Full Network	Application	End-to-End
FINN [126]	FPGA	Compiler	Template	✓	✗	✗	✓	Gen. Purpose	✗
hls4ml [130]	FPGA	Compiler	Template	✓	✗	✗	✓	Particle Colliders	✓
aie4ml [137]	AIE	Compiler	Template	(✓)	✗	✗	✓	Particle Colliders	✗
GNNBuilder [131]	FPGA	Compiler	Template	(✓)	✓	✗	✓	Datacenter	✗
FlowGNN [132]	FPGA	Compiler	Template	(✓)	✓	✗	✓	Datacenter	✗
MaxEVA [133]	AIE	Compiler	Template	✗	✗	✗	✗	Gen. Purpose	✗
AMA [135]	AIE	Compiler	Template	✗	✗	✗	✗	Gen. Purpose	✗
GAMMA [136]	AIE	Compiler	Template	✗	✗	✗	✗	Gen. Purpose	✗
cgra4ml [138]	CGRA	Compiler	Template	✗	✗	✗	✓	Gen. Purpose	✗
ScaleHLS [139]	FPGA	Compiler	Fine	✗	✗	✗	✓	Gen. Purpose	✗
BraggHLS [141]	FPGA	Compiler	Fine	✓	✗	✗	✓	Particle Colliders	✗
ARIES [144]	AIE	Compiler	Fine	✗	✗	✗	✓	Gen. Purpose	✗
mlir-air [145]	AIE	Compiler	Fine	✗	✗	✗	✓	Gen. Purpose	✗
ML-CGRA [142]	CGRA	Compiler	Fine	✗	✗	✗	✗	Gen. Purpose	✗
GRIP [103]	ASIC	Accelerator	App.	✗	✓	✗	✓	Gen. Purpose	✗
PIMGCN [104]	ASIC	Accelerator	App.	✗	✓	✗	✓	Gen. Purpose	✗
BoostGCN [105]	FPGA	Accelerator	App.	✗	✓	✗	✓	Datacenter	✗
StreamGCN [107]	FPGA	Accelerator	App.	✓	✓	✗	✗	Datacenter	✗
H-GCN [108]	FPGA + AIE	Accelerator	App.	✗	✓	✗	✗	Datacenter	✗
Huang et al. [111]	FPGA	Accelerator	App.	✓	✓	✗	✓	Particle Colliders	✗
Que et al. [114]	FPGA	Accelerator	App.	✓	✓	✗	✓	Particle Colliders	✗
Odagiu et al. [113]	FPGA	Accelerator	App.	✓	✓	✗	✓	Particle Colliders	✗
Iiyama et al. [78]	FPGA	Accelerator	App.	✓	(✓)	✗	✓	Particle Colliders	✗
PointAcc [115]	ASIC	Accelerator	App.	✗	(✓)	✓	✓	Automotive	✗
Point-X [116]	ASIC	Accelerator	App.	✗	(✓)	✓	✓	Automotive	✗
Mondal et al. [117]	ASIC	Accelerator	App.	✗	(✓)	✓	✓	Gen. Purpose	✗
DeepBurning [118]	FPGA	Accelerator	App.	✗	✓	✓	✗	Datacenter	✗
Golzar et al. [119]	FPGA	Accelerator	App.	✗	(✓)	✓	✓	Datacenter	✗
DGNN-Booster [120]	FPGA	Accelerator	App.	✗	✓	✓	✗	Datacenter	✗
HgPCN [121]	FPGA	Accelerator	App.	✗	(✓)	✓	✓	Automotive	✓
DeePloy	FPGA + AIE	Compiler	Template	✓	✓	✓	✓	Particle Colliders	✓

This page intentionally left blank

Chapter 4

DeePloy

As established in chapter 3, currently no methodology exists to map graph-based PCNs onto FPGA or CGRA fabric for the low-latency, high-throughput applications encountered in large-scale scientific experiments. This chapter introduces DeePloy, a deployment methodology that addresses this gap. DeePloy provides a design flow in which the latency-critical layers of a graph-based PCN, namely graph building and message passing, are decomposed into smaller operators whose deployment onto hardware modules, the Processing Elements (PEs), can be optimised individually. The flow is deliberately manual, as the design space and the latency constraints of the target applications demand expert judgement at each mapping decision. As the principal conceptual contribution of this thesis, the methodology forms the foundation for the case studies presented in chapters 5 to 7.

This chapter makes the following contributions:

- a deployment methodology for mapping graph-based PCNs onto heterogenous SoCs targeting FPGA and CGRA fabric;
- an architecture template library of PEs for use in conjunction with this deployment flow, supporting both FPGA and CGRA fabric;
- a low-latency sparsity compression approach, compatible with the general design flow, which can be used in conjunction with PCNs mapped with DeePloy.

The remainder of this chapter is structured as follows. Section 4.1 gives an overview of the general methodology. Section 4.2 introduces various graph building schemes suitable for real-time inference. Mapping of static graph-based PCN layers is covered in section 4.3, whereas mapping of dynamic graph-based PCN layers is covered in section 4.4. Section 4.5 introduces the architecture templates developed in this thesis. Section 4.6 introduces a low-latency sparsity compression module, which can be used in conjunction with PCNs mapped with DeePloy.

4.1 Methodology

This section is based on ref. [Neu26c] and has been extended significantly.

For the deployment of graph-based PCNs onto heterogeneous SoCs for real-time inference, DeePloy is developed. DeePloy is an end-to-end deployment methodology that guides algorithm developers and engineers in delivering inference implementations for low-latency, high-throughput environments, such as large-scale scientific experiments. DeePloy is not an automated code-generation tool but a set of guidelines, recommendations, and optimisation procedures that can be applied to graph-based PCNs in general. In the following, DeePloy is described as an implementation-agnostic design flow.

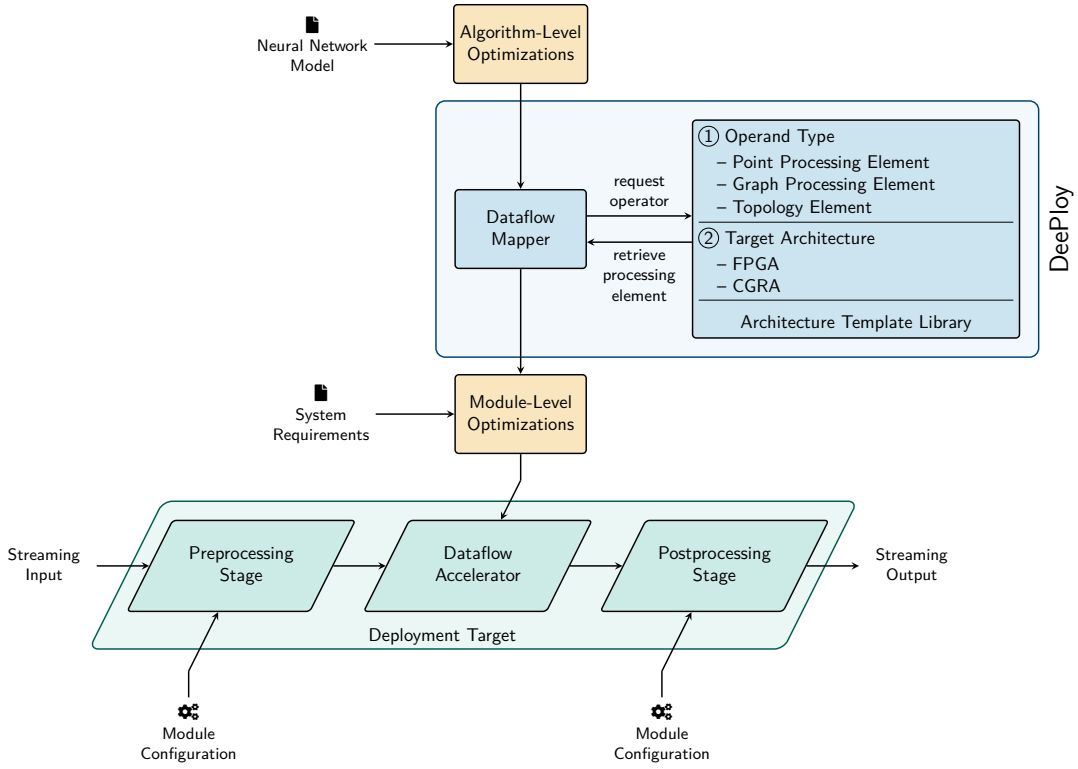


Figure 4.1: Conceptual overview of the DeePloy methodology. The DeePloy core components are shown in blue. The deployment target and its three modules are shown in green. The optimisation stages are shown in orange.

Figure 4.1 gives a conceptual overview of DeePloy. The upper half of the figure depicts the deployment methodology, while the lower half depicts a deployment target. The abstract deployment target assumed in this work is identical for all use cases of DeePloy: streaming data is provided, typically via streaming interfaces such as AXI-Stream [146], Ethernet, or

remote DMA. The preprocessing stage transforms the received packets and extracts input features for the dataflow accelerator. The dataflow accelerator implements the inference pass of a given neural network model. This dataflow accelerator is generated using DeePloy. Finally, the postprocessing stage transforms the outputs of the dataflow accelerator and generates packets suitable for the next stage in the online real-time processing system. Both the pre- and postprocessing stages are typically application-specific implementations. For optimal compatibility with DeePloy, these stages are implemented as parameterised hardware modules that can be configured by supplying a suitable module configuration. Such a module configuration can be generated from the neural network model's interfaces and system requirements.

As input, DeePloy requires a pretrained, quantised neural network model definition, for example, in PyTorch [147], TensorFlow, or Open Neural Network Exchange (ONNX) [148] format, and a set of system requirements. System requirements encompass the following: (1) the target deployment platform, (2) the throughput requirements, and (3) the latency requirements. As a first step, algorithm-level optimisations are incorporated into the model dataflow graph. Two examples of such optimisations are the fusion of layers and operands and the fixed-point approximation of exponential functions. Generally, these optimisations depend strongly on the neural network model.

Next, DeePloy is used to map operands of the neural network onto Processing Elements (PEs) through means of a Dataflow Mapper and an Architecture Template Library. PEs are hardware modules which each map exactly one operand and are fully data-driven. The core of DeePloy is composed of two functional units. First, the Dataflow Mapper requests available operand mappings from the Architecture Template Library and retrieves the PE definitions. Second, the Architecture Template Library contains all available PEs grouped by operand type ① and target architecture ②. It is required that all PEs perform computations in program order, simplifying latency and throughput analysis. Similar to previous work [126, 127, 130], the neural network is modelled as a single-rate dataflow graph. Therefore, all modules represent single-rate actors, meaning the numbers of input and output tokens are identical at all times. To avoid performance degradation, it is enforced that, in all modules, (1) All child pipelines must have the same initiation interval, (2) There must be no stalls at any given time inside the processing element, and (3) The parallelisation must be parameterisable through a parameter. DeePloy defines three types of PEs: (1) Point PEs, (2) Graph PEs, and (3) Topology PEs.

Point Processing Elements perform independent, point-wise computations: each point of an event is processed in isolation, with no dependence on any other point. Point

PEs therefore map operands which are fully spatially separable. One example of such a mapping is dense layers, which are mapped to batched matrix-vector multiplications.

Graph Processing Elements perform relational computations, in which the result for a given point depends on its neighbourhood within the event rather than on that point alone. Graph PEs map graph-based PCN layers composed of a graph building and a message passing step. One example of such a mapping is GravNet layers, which are mapped to a set of PEs in the dataflow accelerator.

Topology Processing Elements do not perform any computation, but facilitate data orchestration between PEs. Topological operators, such as forks and joins, are mapped onto the hardware architecture via these PEs. In addition, changes in hardware parallelisation across different stages of a neural network may be facilitated by topological elements, trading off resource utilisation for throughput and latency.

The output of the mapping step is a single-rate dataflow graph, in which the vertices represent PEs and the edges represent their data communication channels. Based on the provided system requirements, the mapping can now be optimised. The parallelism, and thus the throughput, latency, and system resource utilisation, may be tuned for each PE in the compute graph during the module-level optimisation. If more than one design partition is available on the deployment target, the neural network model may be partitioned based on a cost function. Finally, DeePloy emits Register Transfer Level (RTL) code, HLS C++ code, or C++ code suitable for the respective deployment target. The emitter stage is closely coupled to the actual implementation of the methodology and the definitions of the architecture templates, and is thus not discussed in detail here.

In general, DeePloy allows graph-based PCNs to be deployed with predictable performance as dataflow accelerators, as long as all required operands are available in the Architecture Template Library. Compared with existing works [126, 127, 130], this work focuses specifically on deploying graph-based PCNs and integrating them into an end-to-end deployment target. This focus allows PEs to be developed that run at 100 % pipeline efficiency at all times, maximising throughput and minimising latency for a given configuration. In the following sections, the mapping of graph-based PCN layers onto Graph PEs is described in detail, as this is a central contribution of this work.

4.2 Graph Building

This section is first published in ref. [Neu24a]. It is reproduced here with minimal stylistic modifications.

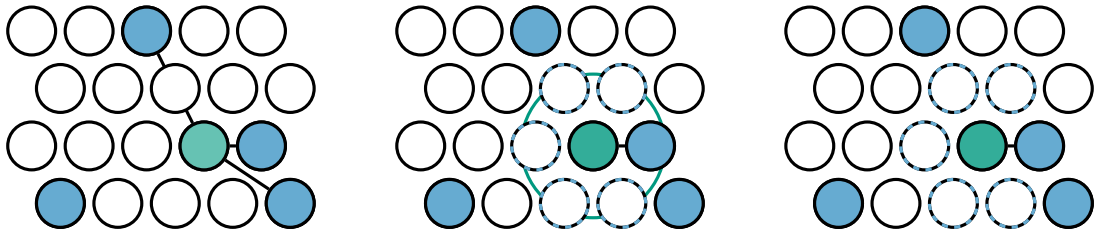
Graph building is a key algorithm in graph-based PCNs (see also section 2.2). This section proposes a methodology for transforming discrete sensor signals captured inside a particle detector into a graphical representation under real-time constraints. It serves as a foundation for the mapping approach for static and dynamic PCN layers.

Current large-scale particle detectors are composed of various discrete sensors and, due to technical limitations, are often placed heterogeneously within the system. For this reason, signals from the sensors cannot be considered regularly distributed, as is the case with, for example, monolithic image sensors. In the following, a detector $\mathcal{V}$ is defined as a set of n discrete sensors $\{\vec{s}_1, \ldots, \vec{s}_n\}$, where each sensor $\vec{s}_i$ is described by a feature vector of length f. Some examples of the described features are the Euclidean location within the detector, the timing information of the received signal, or a discrete *hit identifier*. To map relational connections among individual sensors, a graph based on the detector description is generated, containing the respective sensor features.

Formally described, a graph building algorithm generates an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices of the graph, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. The set of vertices is directly given by the previously described set of sensors in a detector. Each edge $e_{ij} = e(\vec{s}_i, \vec{s}_j) \in \mathcal{E}$ with $\vec{s}_i, \vec{s}_j \in \mathcal{V}$ in the graph connects two sensors based on a building specification that depends on sensor features. The choice of this building specification is a design decision, as it determines both the topology of the resulting graph and the computational effort required to construct it. In the following, the case of building undirected graphs is considered. No fundamental restrictions are introduced that limit the generalisation of this concept to directed graphs.

In general, graph-building approaches are tailored to the specific detector and physics case. Three approaches are considered, which can be classified into two classes of nearest-neighbour graph building: locally constrained and locally unconstrained graphs.

Figure 4.2 depicts an exemplary cut-out of a detector, in which sensors are placed heterogeneously in two-dimensional space. For simplicity, sensors are aligned in a grid-like structure without restricting the generality of the graph-building approach presented



(a) k-NN, for $k = 3$ (b) ε-NN (c) p-NN

Figure 4.2: Example for the three different approaches of building nearest neighbour graphs. Sensors inside a detector are depicted as circles. A sensor which is hit by a particle is blue; those without a hit are white. The query vertices are depicted in green. Solid lines indicate edges connecting the two nearest neighbours. Sensors with dashed outlines are considered candidate sensors, which are part of the specified search pattern around the query vertex. Adapted from ref. [Neu24a].

here. A graph is built for a query vertex, which is depicted by a green circle. The exemplary query vertex is used to illustrate NN-graph building on a single vertex for simplicity. In the following, the three building approaches are compared and their differences explained.

k-NN

k-NN graph building is illustrated on a single query vertex in figure 4.2a. Repeating the building algorithm sequentially leads to a worst-case execution time complexity of $\mathcal{O}(k|\mathcal{V}|\log(|\mathcal{V}|))$ [79]. To reduce execution time, parallelisation of the algorithm has been studied in ref. [149], achieving lower theoretical complexity. Based on the optimisation, a linear $\mathcal{O}(|\mathcal{V}|)$ time complexity is achieved in experimental evaluation [150]. Nevertheless, substantial processing overhead and limitations through exclusive-read, exclusive-write memory interfaces limit the usability for trigger applications. To achieve a higher degree of parallelisation, algorithms described in Refs. [151, 152] use locally constrained approximate graphs.

ε-NN

ε-NN graph building is illustrated on a single query vertex in figure 4.2b. The parameter ε defines an upper bound for the distance of a candidate vertex from the query vertex. All vertices for which equation (4.1) holds true are connected in a graph, yielding a locally

constrained graph. Figuratively, a uniform ball is placed over a query point, joining all edges which are inside the ball into the graph:

$$d(\vec{x}_i, \vec{x}_j) = \|\vec{x}_i - \vec{x}_j\|_2 < \epsilon \tag{4.1}$$

Since the ε-NN approach is controlled by a single parameter, it is a general approach for building location-constrained graphs. However, variations between adjacent sensors in heterogeneous detectors are not well represented in the ε-NN algorithm.

p-NN

Pattern nearest-neighbour (*p*-NN) graph building is illustrated on a single query vertex in figure 4.2c. When building the graph, each candidate sensor is checked, and if a predefined condition is met, the edge between the candidate vertex and the query vertex is included in the graph. The search pattern is defined a priori based on the known detector geometry, so that only a fixed set of candidate sensors in the vicinity of the query vertex is considered.

Comparison

When comparing the *k*-NN, ε-NN, and *p*-NN algorithms, it is apparent that, in general, all three approaches yield different graphs for the same input set of sensors. While the *p*-NN building and the ε-NN building can both be considered locally constrained algorithms, the *k*-NN approach differs as outliers far away from a query point might be included. Nevertheless it is noted in ref. [153], that on a uniformly distributed dataset a suitable upper bound ϵ^* exists, for which the resulting ε-NN graph is a good approximation of the corresponding *k*-NN graph. From a computing perspective, *k*-NN graphs must always be built based on the given input, as graph edges change dynamically depending on the active sensors in the detector. Contrary to ε-NN and *p*-NN graph building, the resulting worst-case graph can be built during compile-time of the neural network. This makes ε-NN and *p*-NN graph-building approaches interesting for static PCN inference, since dynamic scatter and gather operations can be reduced to static switchboxes.

4.3 Mapping Static Point Cloud Networks

This chapter covers the mapping of static graph-based PCN layers onto FPGAs. In section 4.3.1, the previously presented mathematical properties (see section 4.2) are leveraged to demonstrate the feasibility of building approximate k-NN graphs for hardware triggers in large-scale scientific experiments. A method of generating optimised static ε-NN and p-NN graphs from design-time information is presented. This method is used in section 4.3.2 to derive a generic mapping approach for static graph-based PCNs layers, enabling the optimised deployment of such neural networks on FPGAs.

4.3.1 Static Graph Building

This section has been first presented in ref. [Neu24a] and is reproduced here with mininmal stylistic modifications.

The parameter ε in the ε-NN approach and the pattern function in the p-NN approach limit the dimensionality of the graph under construction. Compared with fully connected graphs, the maximum number of edges is reduced by imposing local constraints on sensor connectivity in the detector. Local constraints are implemented by accounting for the influence of static sensor features, such as Euclidean distances between sensors, during the design time of the FPGA firmware. Leveraging a priori knowledge of the sensor position reduces the computational effort required during online inference.

Algorithm 4.1 describes the procedure to derive the intermediate-graph representation of an arbitrary graph-building procedure. As an input, the formally described set of sensors $\mathcal{V}$ is given. Iterating over every sensor in the detector, the proximity of not-yet-visited sensors is assessed using a user-defined *metric* that describes the graph-building approach. If a sensor is considered to be in the neighbourhood of another sensor, the connection is added to the resulting set of edge candidates $\mathcal{E}$. All edges in $\mathcal{E}$ must be checked for their validity during the inference of the online graph building.

The combination of the formal detector description and the set of candidate edges is sufficient to describe an arbitrary building approach on non-directed graphs. According to algorithm 4.1, the worst-case time complexity during design time amounts to $\mathcal{O}(|\mathcal{V}|^2)$, which is higher than the worst-case time complexity of state-of-the-art k-NN building approaches. However, the worst-case runtime complexity now depends only on the number of edges identified during design time. Therefore, generating a graph of low dimensionality by choosing a

suitable *metric* considerably lowers the number of required comparisons at run time. Such an optimisation would not be possible with a k-NN approach, as even for low dimensionality, all possible edges must be considered.

Algorithm 4.1 Design-time graph building

Input: Set of vertices $\mathcal{V}$
Output: Set of edges $\mathcal{E}$

```
1:  procedure BUILDGRAPH(V)
2:      E ← ∅
3:      while V ⊄ ∅ do
4:          s_i ← V.pop()
5:          for all s_j ∈ V do
6:              if metric(s_i, s_j) then
7:                  E ← E ∪ {e_ij}
8:              end if
9:          end for
10:     end while
11:     return E
12: end procedure
```

4.3.2 Layer Formulation

In the following, the general formulation for graph-based PCNs from section 2.2.1 is considered for mapping a single static PCN layer. This approach can be extended without limitation to PCNs with more than one layer, by repeating the mapping for every layer. Figure 4.3 shows a visual overview of the mapping process in four steps.

Figure 4.3a shows an algorithmic-level representation of the PCN layer with set notations. Let $\mathcal{P}$ be the input point cloud and $\mathcal{V} \subseteq \mathcal{P}$ the set of query vertices for the graph-building step. Then let $\mathcal{G}(\mathcal{V}, \mathcal{P}, \mathcal{E})$ be the graph generated in the graph-building step, where $\mathcal{E}$ is the set of edges. In set notation, the message passing step transforms the input point cloud $\mathcal{P} \leftarrow \mathcal{P}'$ to the next layer. This representation is identical to the one described earlier in section 2.2.1.

Figure 4.3b shows an algorithmic-level representation of the PCN layer with matrix notations. The message passing step is decomposed into three high-level operators: Scatter, Transform, and Aggregate. The main difference in this formulation lies in how statically sized matrices are defined. The input point cloud is represented by $P \in \mathbb{R}^{|\mathcal{P}| \times f}$, where f is the feature dimension of each point. The vertex feature matrix is given by $V \in \mathbb{R}^{|\mathcal{V}| \times f}$. Analogously, let

$E \in \mathbb{R}^{|\mathcal{E}| \times d}$ be the edge feature matrix, where d is the feature dimension of each edge. In the following, $e_{i,j}$ denotes a directed edge from $v_i \in \mathcal{V}$ to $p_j \in \mathcal{P}$, and the corresponding row of E is addressed as $E[e_{i,j}, :]$. The message passing step is now performed as follows. First, the scatter operator computes the matrix $S = \mathrm{S}(P) \in \mathbb{R}^{|\mathcal{E}| \times (2f+d)}$ with the following formula:

$$\mathrm{S}(P)[e_{i,j}, :] = P[i, :] \,||\, P[j, :] \,||\, E[e_{i,j}, :], \quad \forall e_{i,j} \in \mathcal{E} \tag{4.2}$$

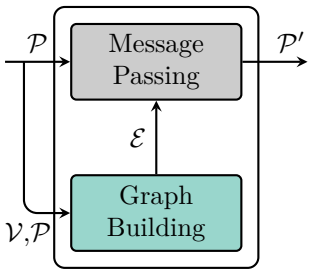
Second, the transform operator computes a message matrix M for every query in the point cloud. This transformation is applied row-wise and is therefore fully spatially parallelisable, in contrast to the scatter and aggregate operators.

Third, the aggregate operator computes the output of the layer P' via a gather function and aggregates the result through permutation invariant functions:

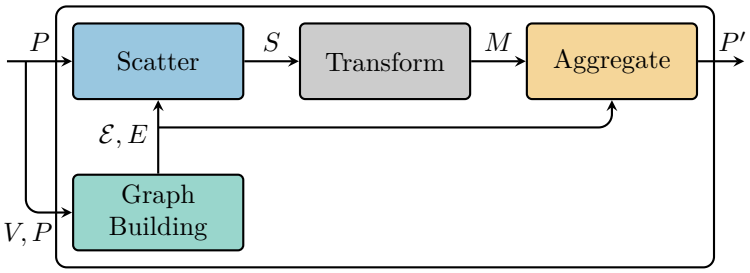
$$\mathrm{G}(M)[i, :] = \bigoplus_{e_{i,j} \in \mathcal{E}} M[e_{i,j}, :] \tag{4.3}$$

Under the assumption that features from P and V that are used for the graph building are constants (known during design time), a series of optimisations can be performed in the inference step of the message passing. Based on the static graph building (see section 4.3.1), indices for the scatter and aggregate operators are known a priori and can be stored in LUTs. Similarly, E is now also a design-time constant and is stored in LUTs. Figure 4.3c shows this transformation. For clarity, scatter is expanded into an atomic scatter and a concat operator.

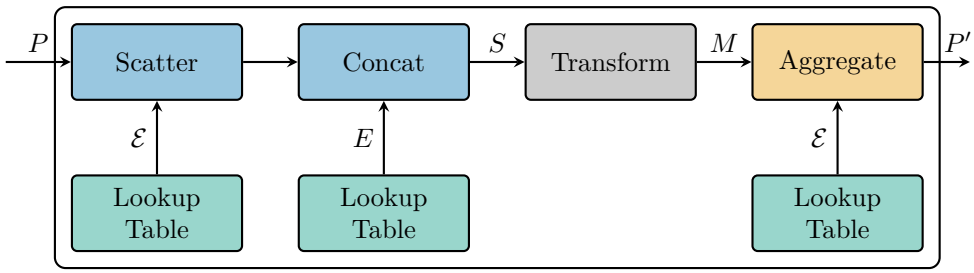
After incorporating the design-time optimisations, all operands are mapped onto their respective PE. During this mapping step, LUTs are merged into the respective modules. The result is a straightforward implementation of a generic static PCN layer, as shown in figure 4.3d. The graph-building step is completed during design time. Scatter and aggregate operators can be realised as static switchboxes: on an FPGA, the fixed addressing scheme can be implemented using wires and multiplexers.



(a) Algorithmic-level representation with set notations.



(b) Algorithmic-level representation with matrix notations. Message Passing is decomposed into Scatter, Transform, and Aggregate.



(c) Algorithmic-level representation with matrix notations. Static graph building is performed during design time.

(d) Module level architecture composed of PEs. PEs are connected via streaming interfaces.

Figure 4.3: Mapping of a single static graph-based PCN layer from set notation on algorithmic level down to a module-level dataflow architecture. Colours indicate logical groups of operands, and PEs over abstraction layers.

4.4 Mapping Dynamic Point Cloud Networks

This chapter covers the mapping of dynamic graph-based PCN layers onto FPGAs. Section 4.4.1 introduces a generic mapping approach that enables the optimised deployment of such neural networks on FPGAs. An abstract architecture is presented in section 4.4.2, serving as a blueprint for architecture templates later in this work.

4.4.1 Layer Formulation

In the following, the general formulation for graph-based PCNs from section 2.2.1 is considered for mapping a single dynamic PCN layer. This approach can be extended without limitation to PCNs with multiple layers by repeating the mapping for each layer. Figure 4.4 shows a visual overview of the mapping process in four steps.

Figure 4.4a shows an algorithmic-level representation of the PCN layer with set notations. Let $\mathcal{P}$ be the input point cloud and $\mathcal{V} \subseteq \mathcal{P}$ the set of query vertices for the graph-building step. Then let $\mathcal{G}(\mathcal{V}, \mathcal{P}, \mathcal{E})$ be the graph generated in the graph-building step, where $\mathcal{E}$ is the set of edges. In set notation, the message passing step transforms the point cloud $\mathcal{P}$ into the next layer's input $\mathcal{P}'$. This representation is identical to the one in sections 2.2.1 and 4.3.2.

Figure 4.4b shows an algorithmic-level representation of the PCN layer with matrix notations. The message-passing step is decomposed into three high-level operators: Scatter, Transform, and Aggregate. The main difference in this formulation lies in how statically sized matrices are defined. The input point cloud is represented by $P \in \mathbb{R}^{|\mathcal{P}| \times f}$, where f is the length of the feature vector of each point. The vertex feature matrix is given by $V \in \mathbb{R}^{|\mathcal{V}| \times f}$. Analogously, let $E \in \mathbb{R}^{|\mathcal{E}| \times d}$ be the edge feature matrix, where d is the length of the feature vector of each edge. In the following, $e_{i,j}$ denotes a directed edge from v_i to v_j, and the corresponding row of E is addressed as $E[e_{i,j}, :]$. The message passing step is now performed as follows. First, the scatter operator computes the matrix $S = \mathrm{S}(P) \in \mathbb{R}^{|\mathcal{E}| \times (2f+d)}$ with:

$$\mathrm{S}(P)[e_{i,j}, :] = P[i, :] \,||\, P[j, :] \,||\, E[e_{i,j}, :], \quad \forall e_{i,j} \in \mathcal{E} \tag{4.4}$$

The graph-building step is decomposed into two high-level operators: All Nearest Neighbour (ANN) and Filter. First, the ANN operator computes the distance of all queries $\mathcal{V}$ to all points $\mathcal{P}$ as a distance matrix $D \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{P}|}$, depending on the user-defined graph

building metric. Second, the filter operator reduces the fully connected graph computed by the ANN operator, depending on the graph building scheme (k-NN, ε-NN, or p-NN).

Figure 4.4c shows the optimised formulation. The gather operation, originally part of the aggregate operator, is merged into the scatter operator. This is possible because both operators are governed by the same edge set $\mathcal{E}$: Scatter distributes point features along the edges, while gather collects messages along the same edges back to their query vertices. Instead of materialising the flat message matrix $M \in \mathbb{R}^{|\mathcal{E}| \times f'}$ and gathering it afterwards, the scatter operator directly produces a tensor $S' = \mathrm{S}'(P) \in \mathbb{R}^{|\mathcal{V}| \times k \times (2f+d)}$, where k is the worst-case upper bound of adjacent neighbours per query vertex (k for k-NN, $|\mathcal{P}|$ for ε-NN and p-NN). The scatter operator writes each edge to a dense, query-aligned address (i, n), where i identifies the query vertex and $n \in \{0, \dots, k-1\}$ is the local slot enumerating the edges incident to v_i.

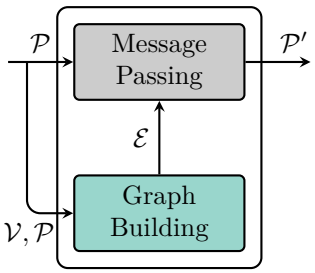
$$\mathrm{S}'(P)[i, n, :] = P[i, :] \,\|\, P[j, :] \,\|\, E[e_{i,j}, :], \quad \forall e_{i,j} \in \mathcal{E} \tag{4.5}$$

The transform operator is applied row-wise over the $|\mathcal{V}| \times k$ message slots, yielding $M' \in \mathbb{R}^{|\mathcal{V}| \times k \times f'}$. As S' is already grouped by query vertex, the aggregate operator reduces over the second axis:

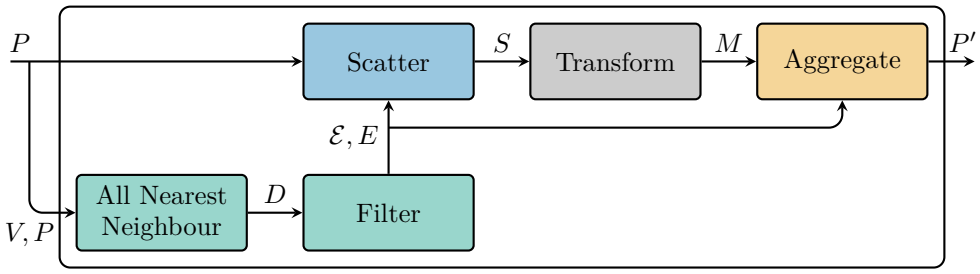
$$\mathrm{G}(M')[i, :] = \bigoplus_{n \in \{0, \dots, k-1\}} M'[i, n, :] \tag{4.6}$$

After incorporating the design-time optimisations, all operands are mapped onto their respective PE. The result is a straightforward implementation of a generic dynamic PCN layer, as shown in figure 4.4d. For clarity, the graph building PE merges ANN and filter operators.

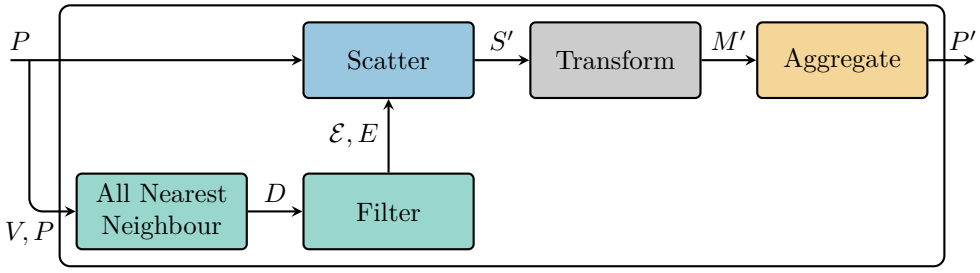
In comparison to the static formulation from section 4.3.2, these optimisations introduce overhead, as the transform operators are now evaluated $|\mathcal{V}| \times k$ times instead of $|\mathcal{E}|$ times. However, the regularity of memory accesses is improved significantly, as only one operator with a dynamic memory access pattern remains. As the graph-building step cannot be completed during design time, the critical path delay of the module is generally longer for a dynamic PCN layer than for a static PCN layer. However, the dynamic PCN layer realises k-NN graph building in general, or repeated graph-building operations on dynamic input features.



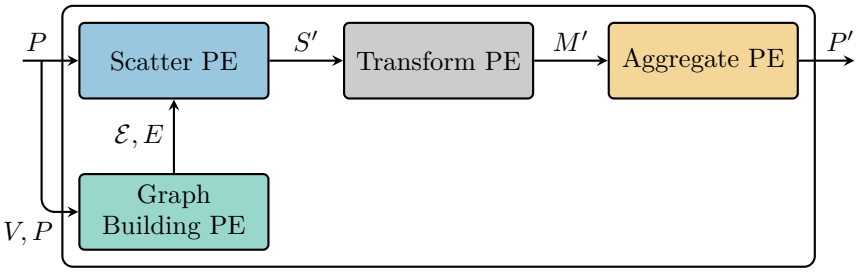
(a) Algorithmic-level representation with set notations.



(b) Algorithmic-level representation with matrix notations. Message passing is decomposed into Scatter, Transform, and Aggregate. Graph building is decomposed into ANN and Filter.



(c) Optimised formulation: fuse the gather operation from Aggregate into Scatter.



(d) Module level architecture composed of PEs. PEs are connected via streaming interfaces.

Figure 4.4: Mapping of a single dynamic graph-based PCN layer from set notation on algorithmic level down to a module-level dataflow architecture. Colours indicate logical groups of operands, and PEs over abstraction layers.

4.4.2 Abstract Architecture

Parts of this section have been first presented in ref. [Neu24b].

To provide a clearer overview of what deploying a dynamic PCN layer as a dataflow accelerator on an FPGA looks like, an abstract architecture is presented below. Figure 4.5 gives a module-level overview of the dynamic PCN layer after the mapping transformation previously described in section 4.4.1. For completeness with the message passing formulation in section 2.2.1, transform operators are added at the input and outputs of the PCN layer. Transform operands ($\beta, \sigma, \phi, \gamma$) are mapped to Transform PEs. In addition, a feedforward path is added. The message passing step is mapped as follows: First, the Scatter PE (compare figure 4.4d) stores all input features into Block Random-Access Memory (BRAM). Based on the edge information supplied by the graph building PE, vertex-level information is distributed to the Transform PEs. In comparison to previous works on GNN accelerators, we decouple edge-level transformation and provide individual BRAMs for each PE [131]. Second, the Transform PEs implement the user-defined message passing function ϕ. Third, edge-level information is aggregated into vertex-level information through the Aggregate PEs. The output features are concatenated with the input path before being transformed in a final step. As stated, the edge information is provided by the Graph Building PE. An overview of the abstract Graph Building PE is given in figure 4.6, exemplary for a k-NN graph building: A query-based approach is used to calculate the nearest neighbours for each point. In the presented version, each point is selected as a query exactly once. The corresponding features are then loaded from RAM and fed into the distance units, which calculate the user-defined distances between the query and the respective candidate nodes. The resulting unsorted distances are provided via streaming interfaces to the subsequent filter PEs. These subsequently sort distances using top-k sort algorithms similar to [154]. By combining bitonic sorts with a recursive merge tree, a single output stream is generated. At this point, it is noted that other sorting algorithms, such as insertion sort, are also applicable [Neu24b]. Finally, a dispatcher generates the respective edges and distributes them to the Scatter PEs as depicted in figure 4.5.

Figure 4.5 and figure 4.6 depict spatial parallelism for accelerating the inference of dynamic PCN layers. For example, two input streams are supplied to the given architecture and processed by two functional groups (Scatter, Transform, Aggregate). Spatial parallelism is applied to the Concat, yielding four output streams. Other examples include the parallelism in the Graph Building PE, where multiple distance units operate in parallel. In practice, all these parallelism factors can be chosen freely.

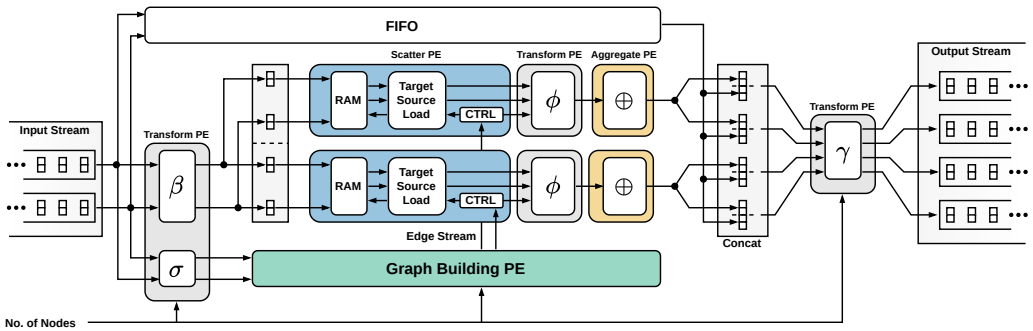


Figure 4.5: Abstract architecture template, depicting a single dynamic PCN layer, including additional transform layers at the beginning and the end of the layer. Adapted from ref. [Neu24b].

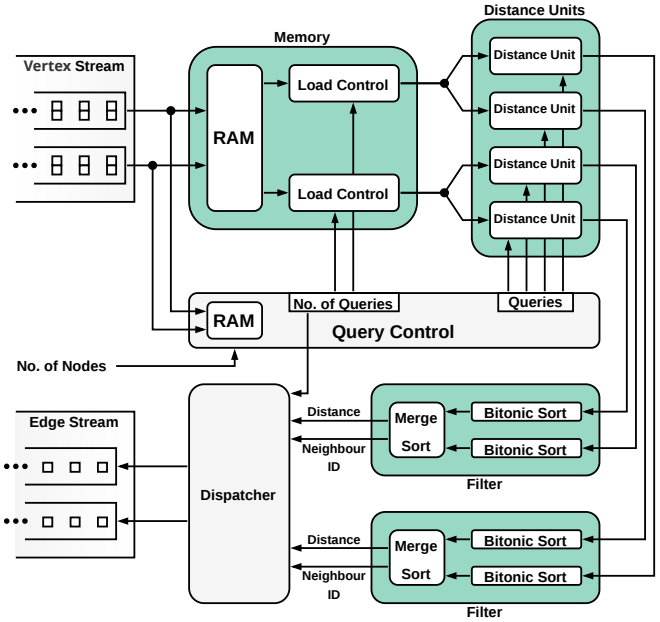


Figure 4.6: Abstract architecture template for a Graph Building PE. This module generates a stream of edges, which is required for the message-passing architecture shown in figure 4.5. Adapted from ref. [Neu24b].

4.5 Architecture Template Library

To support the previously presented mapping schemes for static graph-based PCNs (see section 4.3) and dynamic graph-based PCNs (see section 4.4), I develop an architecture template library. Architecture templates describe instantiable modules which map a specific operator onto the deployment target. In DeePloy, architecture templates are implemented for two different targets: for FPGAs and for CGRAs. For FPGA deployment targets, PEs are either implemented in Chisel [155] or AMD Vitis HLS [156]. As a commercially available, representative CGRA deployment architecture, AMD AIEs are chosen. Architecture templates for AMD AIEs are implemented in AMD Vitis [156] using the proprietary chess compiler. Table 4.1 shows the most important supported operators in DeePloy. For clarity, the three distinct categories for PEs are used: Node, Topology, and Graph PEs (see also section 4.1).

Table 4.1: Overview of the most important supported operators in DeePloy. For each operator, the corresponding PE type is provided, along with the supported deployment targets and the implementation language. As CGRA target, the first generation of AMD AI Engines are chosen. ✓ =supported, (✓) = implicitly supported by toolchain, ✗ =not supported.

PE Type	operator	Target		Implementation Language		
		FPGA	CGRA	Chisel	Vitis HLS	Vitis AIE
Node	Dense	✓	✓	✗	✓	✓
	Linear	✓	✓	✗	✓	✓
	ReLU	✓	✓	✗	✓	✓
	Sigmoid	✓	✗	✗	✓	✗
Topology	Multicast	✓	✓	✓	✓	(✓)
	Split	✓	✓	✗	✓	✓
	Concat	✓	✓	✓	✓	✓
	Tile	✓	✗	✗	✓	✗
	Retile	✗	✓	✗	✓	✓
	Detile	✓	✗	✗	✓	✗
Graph	Static Scatter	✓	✗	✓	✗	✗
	Static Aggregate	✓	✗	✓	✗	✗
	GravNet	✓	✗	✗	✓	✗
	Condensation Point Selection	✓	✗	✗	✓	✗

The Linear operator implements a batched linear transformation in the general form $Y = X \times W + B$, where X is the input matrix and Y is the output matrix. W is the design-time constant weight matrix and B is the design-time constant bias matrix. On FPGA, input and

output vectors are supplied as first-in, first-out (FIFO) streams, and the multiplication kernel is completely unrolled to minimise the latency. This approach is identical to previous works such as FINN [126, 127] or hls4ml [130]. On AMD AIEs, the linear operator is implemented as a batched matrix-matrix multiplication based on the design example provided by AMD [157].

The ReLU operator implements the activation function $\vec{y} = \max(0, \vec{x})$, where $\vec{x}$ is the input vector and $\vec{y}$ is the output vector. On FPGA, the ReLU is typically implemented in distributed LUTs. On AMD AIEs, the ReLU is performed by the Shift-Round-Saturate Unit on the AIE.

The Dense operator fuses the Linear operator and the ReLU operator into a single PE, reducing the overall latency. On FPGA, the overall latency is reduced by two clock cycles, because the FIFO interface at the boundary of the operators is removed. On AMD AIEs, adding the ReLU operation to an existing linear operator comes effectively for free. As the Shift-Round-Saturate Unit performs the ReLU operator on the AIE, this operator can be performed at zero overhead.

The Sigmoid operator implements $y = \sigma(x) = (1 + e^{-x})^{-1}$ in a linearized approximated form [130]. An exact implementation using floating-point arithmetic is unsuitable for low-latency real-time inference, as floating point arithmetic is not vectorized. Therefore, this operator is only available on FPGA targets.

The Multicast operator distributes data by copying the streaming connections on chip. Effectively, this increases the number of concurrent read ports on a given memory element. On FPGA, this operator replicates the data streams. A small overhead is introduced by control-flow logic and greater fanout on datapath nets. On AMD AIEs, this operator is implicitly supported by AMD Vitis and is automatically instantiated by the tool. However, some limitations apply for its usage; multicasting is only supported between AIE tiles which are not adjacent, as the DMA engine instantiation otherwise fails [157]. DMA engines distribute copies of the data from a single source tile into multiple target tiles.

The Concat operator fuses two tensors into a single tensor along the innermost dimension. Due to the tiling and data layout applied in DeePloy, concatenating along the innermost dimension is most efficient in terms of latency and resource utilisation. This operator is supported on both deployment targets.

The Split operator performs the inverse operation to the Concat operator. It is also available for both deployment targets.

Tile, Retile, and Detile operators are required for interfacing between FPGA and AMD AIEs, when Dense or Linear layers are used. The reason lies in the way matrix-matrix multiplications are performed on the AIE. Generally, AMD AIEs require tiled input and output matrices for an efficient inference [157]: in the case of the first generation of AMD AIEs, the input matrix X must be tiled in (8×4) layout and the weight matrix must be tiled in (4×8) layout. The bias matrix and output matrix B, Y must be tiled and zero-padded as (4×4). The underlying architecture templates for Tile, Retile, and Detile are generally applicable to all two-dimensional tiling schemes, enabling forward compatibility with newer AMD AIE generations. On FPGA, Tile and Detile are available, converting row-wise matrices into tiled matrices and vice versa. On AMD AIEs, Retile is available to convert from one tiled format to another. Usually, this is used to daisy-chain multiple dense layers.

4.5.1 Static Scatter

An overview of the architecture template implementation is given in algorithm 4.2. The PE receives the point cloud feature matrix P as an input, as well as the edge feature matrix E via parallel streaming interfaces. Both matrices have static sizes determined by the design-time graph building step (see also section 4.3.1). As a design-time parameter, the static edge list $\mathcal{E}$ is provided, which enables the construction of a mapping from vertices to edges. The function active checks if the given edge is active in the current inference step. The PE provides a streaming interface with the output scatter matrix S. Each row of S corresponding to an active edge is formed by concatenating the source vertex features, the destination vertex features, and the edge features.

Due to the statically bounded size of all matrices and the edge list, the for loop in the algorithm definition can be fully unrolled. Increasing the parallelism of the PE increases the number of read and write ports linearly, without incurring any penalty in time or space complexity. This makes the formulation particularly suitable for low-latency real-time applications. The mapping of the Static Scatter operator to its PE is previously described in section 4.3.2. For a description of a complete static PCN, see section 2.2.2.

4.5.2 Static Aggregate

An overview of the architecture template implementation is given in algorithm 4.3. The PE receives the message matrix M from the previous transform PE via a parallel streaming interface. The static adjacency matrix A is calculated during compile-time and stored in LUT

Algorithm 4.2 Architecture template for the Static Scatter operator.

Require: point cloud feature matrix $P \in \mathbb{R}^{|\mathcal{P}| \times f}$
Require: edge feature matrix $E \in \mathbb{R}^{|\mathcal{E}| \times d}$
Require: static edge list $\mathcal{E}$, where each $e_{i,j}$ is a directed edge from v_i to p_j
Ensure: scatter matrix $S \in \mathbb{R}^{|\mathcal{E}| \times (2f+d)}$

parallel for $e_{i,j} \in \mathcal{E}$ **do**
  if active(v_i, p_j) **then**
    $S[e_{i,j}, :] \leftarrow P[i, :] \,||\, P[j, :] \,||\, E[e_{i,j}, :]$
  else
    $S[e_{i,j}, :] \leftarrow \mathbf{0}$
  end if
end for
return S

on FPGA. In this particular case, k denotes the maximum degree in the graph $\mathcal{G}(\mathcal{V}, \mathcal{P}, \mathcal{E})$ built during compile-time (see also section 4.3.1). As adjacency features are stored in distributed LUTs, empty elements in the matrix do not consume any system resources; elements which are never written or read are optimised by the synthesis tool. The outer loop over the $|\mathcal{V}|$ vertices can be fully parallelised, as each output row $P'[i, :]$ is computed independently. For each vertex, the k candidate edges are processed by a parallel reduction: the validity predicate valid$(M[e, :])$ gates each lane, so that only edges carrying a valid message contribute, and the remaining messages are combined by the permutation-invariant aggregate function. Both valid and aggregate are template parameters of the operator, bound at instantiation time to the concrete predicate and reduction function required by the respective PCN layer. On the hardware side, this reduction is implemented as a balanced reduction tree over the k lanes, yielding latency logarithmic in k while keeping the fan-in statically bounded.

For clarity, it is noted that this architecture template keeps aligned to the formulation in section 4.3.2 and does not fuse subsequent concatenations or requantization steps into the PE. The mapping of the Static Aggregate operator to its PE is previously described in section 4.3.2. For a description of a complete static PCN, see section 2.2.2.

4.5.3 GravNet

Parts of this section have been adapted from ref. [Neu26a].

As a concrete instantiation of the abstract architecture template introduced in section 4.4.2, the mapping of the GravNet operator as a dedicated PE is presented in the following. This PE

Algorithm 4.3 Architecture template for the Static Aggregate operator.

Require: message matrix $M \in \mathbb{R}^{|\mathcal{E}| \times d}$
Require: static adjacency matrix $A \in \mathbb{N}^{|\mathcal{V}| \times k}$, where $A[i,m]$ is the m-th incident edge of v_i
Ensure: output matrix $P' \in \mathbb{R}^{|\mathcal{V}| \times d}$

parallel for $i \leftarrow 0$ **to** $|\mathcal{V}| - 1$ **do**
  $acc \leftarrow \mathbf{0}$
  parallel for $m \leftarrow 0$ **to** $k - 1$ **do**
    $e \leftarrow A[i,m]$
    if valid$(M[e,:])$ **then**
      $acc \leftarrow$ aggregate$(acc, M[e,:])$
    end if
  end for
  $P'[i,:] \leftarrow acc$
end for
return P'

combines the graph building and message passing steps of a single GravNet layer, following the generic mapping methodology described in section 4.1 and the layer formulation derived in section 4.4.1. For a description of the full GravNet layer, see section 2.2.3. An overview of the architecture template is given in figure 4.7.

As an input, the PE receives two input streams. The first input stream connects to the ANN operator, transmitting points in the learned latent space of the GravNet layer. The second input stream connects to the scatter operator, transmitting points in the learned feature space of the GravNet layer. The graph-building step is realised through an ANN operator followed by a hierarchical top-k sort, facilitating a dynamic k-NN graph-building approach. For the message-passing step, the sorted features are retrieved from the BRAM ping-pong buffers and multiplied by exponentially weighted distances using dedicated exponentiation and multiplication operators. Neighbourhood aggregation is subsequently performed through max-reduce and sum-reduce operators. Finally, the output features are aggregated into a single streaming interface.

4.5.4 Condensation Point Selection

Parts of this section have been adapted from ref. [Neu26a].

As a further instantiation of the abstract architecture template introduced in section 4.4.2, the mapping of the Condensation Point Selection (CPS) operator onto a dedicated PE is presented in the following. In contrast to the GravNet operator described in section 4.5.3,

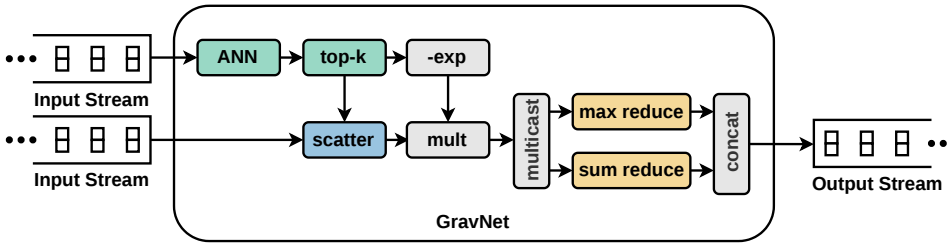


Figure 4.7: Architecture template for the GravNet operator. The feedforward path and linear operators are omitted for clarity, compared with the layer definition in section 2.2.3. Adapted from Refs. [Neu26a, Neu25b].

the CPS operator does not constitute a point cloud layer with message passing. Instead, it implements a clustering algorithm operating in the latent space created by previous GNN layers. For a complete description of the CPS algorithm, see also section 2.2.4. An overview of the architecture template is given in figure 4.8.

The isolation between cluster seeds is first determined through an ANN operator, followed by an isolation selection. The combination of the ANN operator and the Isolation Selection effectively implements an ε-NN graph-building step. Candidate cluster points are masked by a β threshold during candidate selection and then ordered by a bitonic sort. As a last step, the Condensation Point Candidate Selection (CPCS) is applied.

In the following, the implementation of the CPCS algorithm is described in detail, as it constitutes the core compute step of the CPS. The corresponding pseudo-code is given in algorithm 4.4. It computes the subset of clusters from n respective condensation point candidates. Condensation point candidates are expected to be stored strictly in order, such that each candidate is uniquely identified by its memory address (*id*). The unit requires the following two inputs for every query candidate q_i: (1) a bitmask $\{0,1\}^n$ containing the isolation criteria between q_i and all other candidates, and (2) a boolean flag indicating whether the β criterion is fulfilled for the query candidate q_i. These values are stored in the arrays *isolations* and *candidates*. In addition, an ordered list of query *ids* is required to ensure that query candidates with higher β values are prioritised. The output is a bitmask $\{0,1\}^n$ indicating which query candidates at the corresponding memory locations have been selected as cluster condensation points.

The simplicity of the implementation is emphasised, as the computationally intensive steps have been offloaded to previous pipeline stages within the CPS PE. As a result, the realisation of the condensation point clustering selects up to p_{par} clusters per clock cycle, yielding an initiation interval of $ii = \lceil \frac{n}{p_{\text{par}}} \rceil$. Notably, no limitation exists on the number of clusters to be

selected, provided that p_{par} is chosen appropriately. The algorithm can be statically pipelined and thus complies with hard real-time requirements.

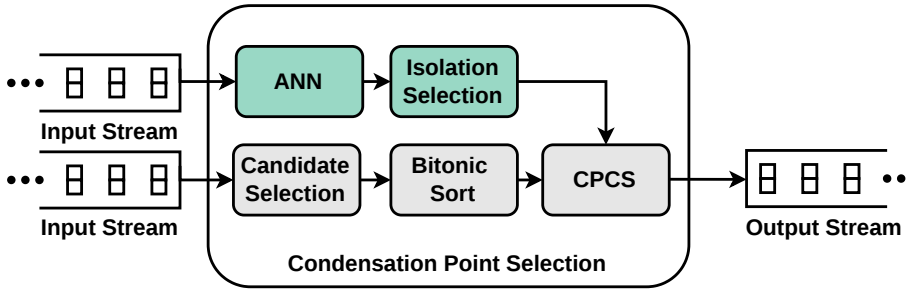


Figure 4.8: Architecture template for the Condensation Point Selection operator. Adapted from Refs. [Neu26a, Neu25b].

Algorithm 4.4 Condensation Point Candidate Selection (CPCS). Taken from ref. [Neu26a].

```
1:  procedure CPCS(isolations,candidates,ids)
2:      cps ← {0}^n
3:      flags ← candidates
4:      for i ← 0, ii − 1 do
5:          parallel for p ← 0, p_par − 1 do
6:              id ← ids.pop()
7:              cps[id] ← flags[id]
8:              flags ← flags & isolations[id]
9:          end for
10:     end for
11:     return cps
12: end procedure
```

4.6 Sparsity Compression

This section is based on ref. [Neu25b] with minor stylistic modifications.

The preprocessing stage introduced in figure 4.1 is, in general, an application-specific implementation. Nonetheless, recurring functionality can be encapsulated in reusable, parameterised hardware modules that are instantiated according to a suitable module configuration. In this section, the *stream compaction module* is presented as a reusable component that addresses the structured sparsity inherent in point cloud data from a detector front-end.

Consider the point cloud from equation (2.1). A detector consists of n_{tot} sensors, where the data of each unique sensor is represented by a feature vector that represents $\vec{p}$ in $\mathbb{R}^f$. Thus, an event is defined as a point cloud $\mathcal{P} = \{\vec{p}_0, \ldots, \vec{p}_{n-1}\} \subseteq \mathbb{R}^f$ of n points with dimension f, where $n \in \mathbb{N} : n \leq n_{tot}$. The number of points n, and therefore the input sparsity for a given event, can be adjusted through the sensor sensitivity, i.e. the energy threshold. Since the detector is fully controlled, the energy threshold is chosen so that an upper bound $\overline{n}$ on the number of sensors that can simultaneously be hit in a given event can be derived. If an event contains more than $\overline{n}$ hits, additional points are dropped from the input set. By this definition, hard real-time requirements are met if the accelerator can process up to $\overline{n}$ sensors simultaneously hit in a given event. The actual number of points n for a given event is unknown a priori and is thus treated as an additional input to the network.

Generally, the sensor data for each event is represented in a matrix $X_{tot} \in \mathbb{R}^{n_{tot} \times f}$. However, due to the variable number of points per event, this matrix contains a large number of zero rows, i.e. structured sparsity. To process such matrices efficiently, either a suitable dynamic sparse tensor operator or a compaction mechanism that exploits the sparsity pattern is required. The latter option is chosen because the worst-case execution time of dynamic operators is difficult to predict. Thus, for each event, X_{tot} is replaced with a compacted matrix $X \in \mathbb{R}^{\overline{n} \times f}$. To retain the original position of each point in X_{tot}, an additional matrix $Y \in \mathbb{N}^{\overline{n} \times 1}$ is introduced, containing the original index of each non-zero point in the array. The dataflow accelerator generated by DeePloy expects the matrices X and Y and the scalar n as inputs for a given point cloud network to compute the inference result Z.

In the literature, sparsity-aware stream compression has been considered for database applications [158, 159] as well as for near-sensor bandwidth compression of time-series data for scientific applications [160]. However, these works focus on minimising bandwidth overhead

between platforms rather than providing a preprocessing stage for neural network inference. In the approach presented here, a large number of parallel interfaces carrying sparse data are reduced to a smaller number of parallel interfaces, with the intention of densifying the streams on the same compute platform and maximising subsequent utilisation across pipeline stages.

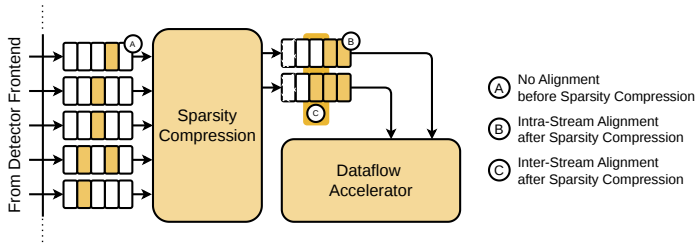
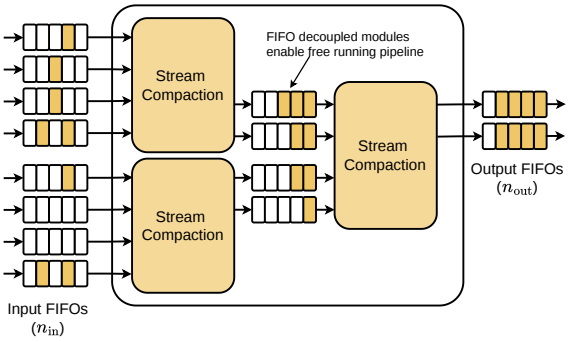


Figure 4.9: Typical application of the sparsity compression module. Sparse data are provided by the detector frontend and compacted for subsequent processing in a dataflow accelerator. Adapted from ref. [Neu25b].

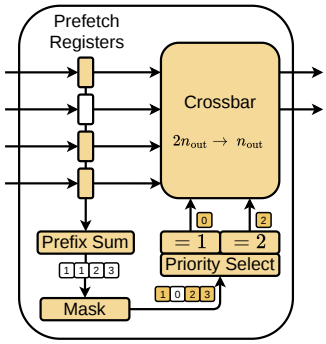
A conceptual top-level system diagram of the approach is shown in figure 4.9. The sparsity compression module receives n_{in} FIFO interfaces as input ports and provides n_{out} FIFO interfaces as output ports. In every clock cycle, each input port receives one data element. Data elements from up to c consecutive clock cycles are processed in a single neural network inference. Non-zero data elements are coloured yellow, whereas empty data elements are white.

The sparsity compression module fulfils two functions, intra- and inter-stream alignment. For the former, the sparse input data Ⓐ is rearranged into dense streams Ⓑ. For the latter, the dense streams are balanced Ⓒ. The following explains how this functionality is achieved.

The sparsity compression module depicted in figure 4.10a is composed of stream compaction modules. Each cell inside the hierarchical tree compacts $2 \cdot n_{out}$ input ports into n_{out} output ports. All cells are connected via FIFO interfaces in a tree topology. The number of stages depends on the ratio between the input and output ports of the sparsity compression module. The stream compaction module depicted in figure 4.10b is the central module of the sparsity-compression approach. Its three pipeline stages work as follows: (1) data elements are loaded into prefetch registers; (2) a bitmask is generated from the valid signal, a prefix sum over all valid bits is calculated and applied to the input mask, and two daisy-chained priority encoders select the addresses of the first two non-zero data elements; and (3) based on the previously calculated addresses, the crossbar is configured to forward the data elements to the correct output ports.



(a) Sparsity Compression



(b) Stream Compaction

Figure 4.10: Conceptual block-level diagram of the sparsity compression hardware module. In the exemplary case, the following values are chosen: $n_{in} = 5$, $n_{out} = 2$, and $c = 5$. Adapted from ref. [Neu25b].

As a reusable preprocessing component in the sense of figure 4.1, the module is configured solely through the parameters n_{in}, n_{out} and c, which can be derived from the interfaces of the neural network model and the system requirements. The approach achieves several useful properties: (1) the stream compaction module achieves a deterministic latency as described in equation (4.7); (2) by specifying the depth of the window d statically, the stall-free pipeline enables 100 % utilisation; and (3) routeability of the design on FPGA is retained, as the crossbar size only scales with the ratio of the module ports.

$$l = 3\left\lceil \log_2\left(1 + \frac{n_{in}}{n_{out}}\right)\right\rceil f_{\text{sys}}, \quad n_{in} > n_{out} \tag{4.7}$$

Chapter 5

Graph Neural Network Hit Cleanup at Belle II

This chapter is partly published in refs. [Hei26b, Hei26c, Neu24a].

The current Belle II CDC L1 Trigger is a multi-stage trigger implemented on FPGA, responsible for reconstructing the momentum and position of tracks originating near the interaction point (see also section 2.1.3). To obtain a trigger decision, track segments are generated in parallel from 14 336 sense wires. Based on these track segments, 2D and 3D track-finding algorithms, as well as a neural network [Bäh25], predict track objects with varying precision, efficiency, and purity for the GDL [Lai25]. However, the current trigger faces three challenges: First, with SuperKEKB approaching an instantaneous luminosity of $6 \times 10^{35}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$, rising beam backgrounds increase the number of active sense wires per detector snapshot. As a result, the track segment finders operate outside their intended design point, reducing trigger efficiency. Second, increased occupancy of sense wires makes the crosstalk between adjacent sense wires more prominent. This crosstalk is especially severe for sense wires in the same layer. Third, the CDC suffers from ageing, which leads to further degradation of the CDC L1 Trigger efficiency.

To address these challenges, a prototype of the Graph Neural Network Hit Cleanup Trigger Module (GNN-HCM) has been developed as a hardware trigger on FPGA. GNN-HCM is a case study that considers deploying a quantised, pretrained static PCN in the CDC L1 Trigger. The remainder of this chapter is structured as follows:

Section 5.1 briefly describes the hit-cleanup algorithm and the system requirements. Section 5.2 describes the dataset used for analysing the graph building on the Belle II CDC. Section 5.3 covers the partitioning of the CDC into sectors, the derivation of p-NN-based graph building, and the comparison to k-NN and ε-NN alternatives. Section 5.4 covers the architecture of the GNN-HCM dataflow accelerator. Section 5.5 describes the deployment of the static PCN. Section 5.6 evaluates the system performance in RTL simulation. Section 5.7 discusses the results and concludes the chapter.

5.1 Algorithm and Design Requirements

The development of the algorithm and the training of the neural network have been studied in Refs. [2, Hei26b, Hei26c]. In the following, the model description from ref. [49] is summarised to provide the necessary context for deriving the system requirements.

The GNN-HCM implements a static PCN, based on the Interaction Network architecture (section 2.2.2). This architecture was chosen because GNN-based hit filtering has been successfully applied in offline reconstruction at Belle II [2, 17].

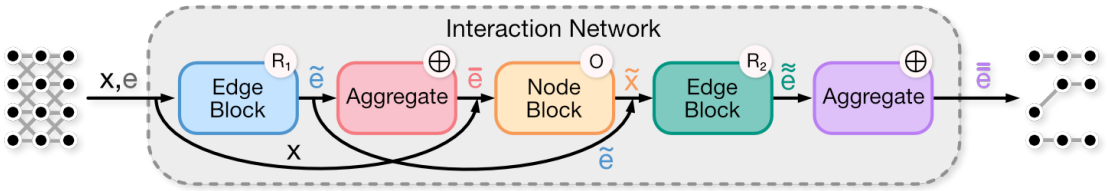


Figure 5.1: Illustration of the Interaction Network developed in Refs. [Hei26b, Hei26c], without static graph building. Edge blocks R and vertex block O are MLPs; aggregation operations are denoted by $\oplus$. Taken from ref. [May25].

Figure 5.1 shows an illustration of the neural network model, excluding the graph building step typically required in static PCNs (see also section 2.2.1). As an input, the model receives trigger sense wires from the CDC detector (section 2.1.3) with four features each: time, amplitude, x, and y coordinates of the sense wire at $z = 0$. The inference can be separated into three steps.

First, a graph $G(\mathcal{V},\mathcal{E})$ is constructed from the input features. The graph is built on a local neighbourhood defined as a pattern (see also p-NN in section 4.2). The output of the graph-building step consists of the graph's vertex and edge features. Per-vertex features $\vec{x} \in \mathbb{R}^2$ contain the amplitude readout (ADC) and layer id of the respective sense wire. Per-edge features $\vec{e} \in \mathbb{R}^3$ contain the differences of polar angle θ, time readout (TDC), and the layer id between adjacent vertices. In a compact form, the output of the graph building step is written as a vertex feature matrix $X \in \mathbb{R}^{2\times|\mathcal{V}|}$ and an edge feature matrix $E \in \mathbb{R}^{3\times|\mathcal{E}|}$.

Second, information is propagated through the Interaction Network in figure 5.1 via a series of message passing layers. Edge blocks R denote MLPs transforming every edge, e.g., the first edge block transforms $\mathsf{R}_1 : e \to \tilde{e}, \forall e \in \mathcal{E}$. Node blocks O denote MLPs transforming graph vertices, e.g., the vertex block transforms $\mathsf{O} : x \to \tilde{x}, \forall x \in \mathcal{V}$. Aggregate blocks $\oplus$ transform data from edge-level representation into vertex-level representation. Scatter

operations are implicitly shown in the figure when transforming from vertex-level to edge-level representation. The output of the Interaction Network inference is a classification score for every edge.

Third, the classification score for each edge is applied to its adjacent vertices. Based on a threshold treated as a runtime hyperparameter, all vertices adjacent to at least one edge whose classification score exceeds this threshold are forwarded to successive trigger stages. Remaining vertices are considered background and subsequently dropped from the event.

The goal of this case study is to evaluate the applicability of GNN-based hit filtering in the Belle II CDC L1 Trigger, motivated by the planned upgrades during Long Shutdown 2 at Belle II. Currently, the CDC L1 Trigger consists of multiple UT 4 hardware boards; The filtering of sense wires is performed by nine so-called Track Segment Finders (section 2.1.3). This work aims to replace these nine Track Segment Finders with GNN-based hit-filtering modules, called GNN-HCM. Due to architectural optimisations elsewhere in the trigger system, it is expected that up to 20 UT 4 hardware boards will be freed for this purpose in the near future. All other system requirements remain similar to those of the filtering stage in the CDC L1 Trigger.

Three system requirements for the GNN-HCM follow from this assumption, the algorithm itself, and its position within the trigger hierarchy:

Requirement 5.1. The GNN-HCM shall deploy the full Interaction Network inference pipeline, including graph building, onto up to 20 FPGA platforms.

Requirement 5.2. The GNN-HCM shall be made available to the successive trigger stages before the readout buffer of the L1 Trigger system overflows. Based on estimations in the current system, a hard real-time deadline of $R_{\mathrm{L}} = 500\,\mathrm{ns}$ is imposed on the end-to-end latency.

Requirement 5.3. The GNN-HCM shall sustain the full input rate of $R_{\mathrm{th}} = 32$ million detector snapshots per second. Each snapshot contains the values of all trigger sense wires; sense wire hits shall be accumulated for 512 ns.

5.2 Dataset

This section is taken from ref. [Neu24a] with minor stylistic changes.

In this work, simulated Belle II events are used to benchmark the graph-building algorithms. The detector geometry and interactions of final state particles with the material are simulated using GEANT4 [161], which is combined with the simulation of a detector response in the Belle II Analysis Software Framework [162]. The Belle II detector consists of several subdetectors arranged around the beam pipe in a cylindrical structure, as described in detail in refs. [5, 24]. The solenoid's central axis is the z-axis of the laboratory frame. The longitudinal direction, the transverse xy plane with azimuthal angle ϕ, and the polar angle θ are defined with respect to the detector's solenoidal axis in the direction of the electron beam. The CDC consists of 14 336 sense wires surrounded by field wires, which are arranged in nine so-called superlayers of two types: axial and stereo superlayers. The stereo superlayers are slightly angled, allowing for 3D reconstruction of the track. In the simulated events, only the detector response of the CDC is retained.

Two muons (μ^+,μ^-) per event are simulated with momentum $0.5 < p < 5\,\mathrm{GeV/c}$, and direction $17° < \theta < 150°$ and $0° < \phi < 360°$ drawn randomly from independent uniform distributions in p, θ, and ϕ. The generated polar angle range corresponds to the full CDC acceptance. Each of the muons is displaced from the interaction point between 20 cm and 100 cm, where the displacement is drawn randomly from independent uniform distributions.

As part of the simulation, simulated beam background events are overlaid corresponding to instantaneous luminosity of $6.5\times10^{35}\,\mathrm{cm^{-2}s^{-1}}$ [163, 164]. The simulated conditions are similar to those expected when the experiment reaches its ultimate luminosity.

5.3 Graph Building for the Central Drift Chamber

This section is adapted from ref. [Neu24a] with minor changes.

I apply the graph-building methodology from section 4.2 to the Belle II CDC L1 Trigger. For the comparison of various graph-building approaches, the dataset described in section 5.2 is used.

The CDC 's wire configuration is mapped to the formal detector definition, with wires serving as discrete sensors. These sensors are referred to as nodes or vertices in the following. Inside the L1 Trigger three signals are received per wire: a hit identifier (HIT), the time readout (TDC) and the amplitude readout (ADC), where TDC is the output of a time-to-digital converter measuring the drift time, and ADC is the output of an analogue-to-digital converter measuring the signal height that is proportional to the energy deposition in a drift cell. Cartesian coordinates of the wires inside the detector are known during design time and used as static sensor features. Additionally, the distance between two vertices, which is also known at design time, is considered an edge feature.

Illustrating the working principle of the graph building approaches, figure 5.3 depicts four cut-outs of the CDC in the x-y plane for $z = 0$.
In sector Ⓐ, hits received by the detector for an exemplary event are indicated by black markers. The other three sectors show one graph building approach each: Sector Ⓑ depicts a k-NN graph for $k = 6$, as there are up to six direct neighbours for each wire. The k-NN graphs connect wires that are widely separated. Sector Ⓒ shows an ε-NN graph for $\varepsilon = 22$ mm. The specific value for ε is chosen, because 22 mm is in the range of one to two neighbour wires inside the CDC. This graph-building approach connects hits only in close proximity, yielding multiple separate graphs. In addition, more edges are detected in the inner rings compared to the outer rings of the detector due to the higher wire density in this region. Finally, sector Ⓓ shows a p-NN graph using the pattern described in figure 5.2. The pattern extends the existing pattern [51, Ung23b, 165] of the currently implemented Track Segment Finder in the CDC L1 Trigger by taking neighbours in the same superlayers into account. It is identical to the pattern used in ref. [49]. When comparing the ε-NN graphs and the p-NN graphs with each other, it is observed that the degrees of p-NN vertices are more evenly distributed (see inserts in figure 5.3).

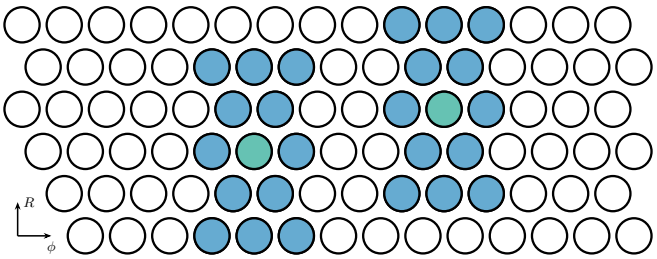


Figure 5.2: Two query vertices illustrate the neighbourhood pattern in the hourglass shape used for the Belle II detector case study. The superlayer is rolled off radially, and an exemplary cut-out is shown. Vertices which are considered neighbour candidates of the respective green query vertex are shown as blue markers. Taken from ref. [Neu24a].

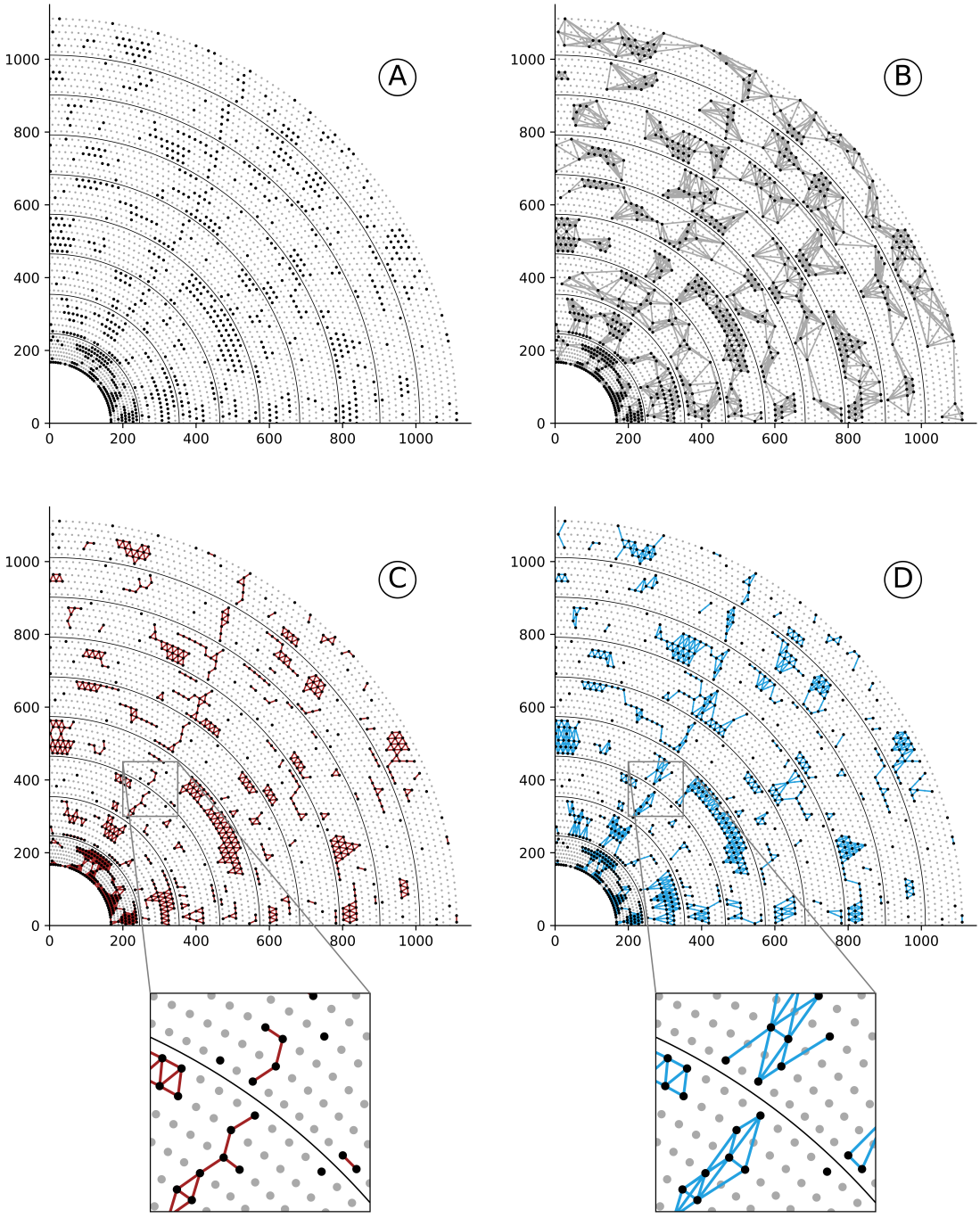


Figure 5.3: Typical event display of the CDC for various graph building approaches. Quadrants show Ⓐ all hits, Ⓑ k-NN graph building (k=6), Ⓒ ε-NN graph building (ε=22 mm), and Ⓓ p-NN graph building (see figure 5.2). The inserts show zooms to a smaller section of the CDC. Taken from ref. [Neu24a].

5.3.1 Parameter Exploration

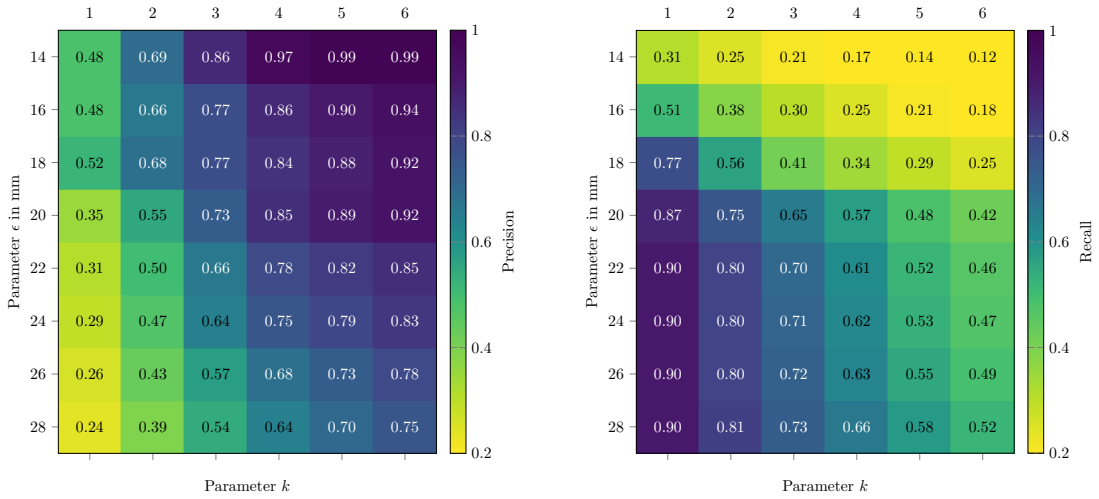
In general, k-NN, ε-NN, and p-NN algorithms generate different graphs for an identical input event. However, to replace k-NN graph building with a locally constrained graph building approach, the graphs should ideally be identical. As the generated graphs depend strongly on the chosen hyperparameters, on the geometry of the detector, and on the hit distribution of the events under observation, a quantitative measure of the similarity of the generated graphs between k-NN graphs and locally constrained graphs, such as ε-NN or p-NN graphs, is necessary. The optimal choice of the hyperparameter ε^* is the one that maximises the similarity for any k. For optimisation, I use simulated events as described in section 5.2. Both the k-NN graphs and the locally constrained graphs are generated from the dataset, considering the neighbourhood of wires within the detector. Edges of the k-NN graphs are labelled $\mathcal{E}_k$, whereas the edges of observed locally constrained graphs are labelled $\mathcal{E}_l$. The similarity between the two graphs is measured using the binary classification metrics, recall, and precision defined as

$$recall = \frac{|\mathcal{E}_k \cap \mathcal{E}_l|}{|\mathcal{E}_k|}, \tag{5.1}$$

$$precision = \frac{|\mathcal{E}_k \cap \mathcal{E}_l|}{|\mathcal{E}_l|}. \tag{5.2}$$

The parameter k is varied between 1 to 6 and ε between 14 mm to 28 mm, as the minimal distance between two wires in the CDC is approximately 10 mm. Precision and recall scores are calculated for every pair of k and ε parameters and show the mean value over 2000 events in figure 5.4. As expected, the precision score increases monotonically as the parameter k increases. In addition, it increases as the parameter ε decreases. The recall score behaves in the opposite way: it decreases monotonically as parameter k increases. In addition, it decreases as the parameter ε decreases. Similarity is defined as the ratio of recall to precision, where an optimal working point maximises both recall and precision. High similarity is not found for all values of k. Maximal similarity is found for $k = 3$ and $\varepsilon = 22$ mm, and $k = 4$ and $\varepsilon = 28$ mm, respectively. The corresponding precision and recall on the underlying data set are around 65-70%.

The similarity between k-NN and ε-NN graphs can be interpreted in relation to the mathematical statement from ref. [153] (compare section 4.2). Based on the background noise and the large number of hits per event, it is assumed that hits in the dataset are approximately uniformly distributed. Therefore, it is expected that pairs of k-NN and ε-NN graphs exist that exhibit a high degree of similarity, e.g. precision and recall scores close to one. This expectation is only partially met as the trade-off point reaches only about 65-70 %. One possible



(a) Precision (b) Recall

Figure 5.4: Precision and recall for the comparison of the k-NN and ε-NN graph building approaches. Values taken from ref. [Neu24a].

reason for the remaining difference between the two graphs is the underlying background noise. Although the events are clearly dominated by noise, their influence on the hit distribution is not strong enough to yield higher similarity scores.

The same comparison is performed between the k-NN and the p-NN graph building approach as shown in figure 5.5. Similar results are achieved in comparison to the ε-NN comparison: The recall score is monotonically decreasing for a larger parameter k, and the precision score is monotonically increasing for larger parameter k. For k between three and four, precision and recall are approximately equal at around 70 %.

Again, this expectation of a high degree of similarity is only partially met. This similarity is to be expected, as the chosen pattern is ellipsoidal in the r–ϕ space.

5.3.2 Sector Partitioning

For implementing the proposed algorithm in a hardware prototype, the CDC is partitioned into 20 sectors in ϕ and radial distance r for the L1 Trigger. Sectors overlap in ϕ, such that shallow tracks are not cut-off artifically by the segmentation. Overlapping sectors in r are not realized in the hit-cleanup stage, as this would require modification of the CDC frontend electronics. The assignment of the wires to the individual hardware modules is based on the associated position and the azimuthal angle ϕ in the CDC. Each ϕ-r-sector is processed

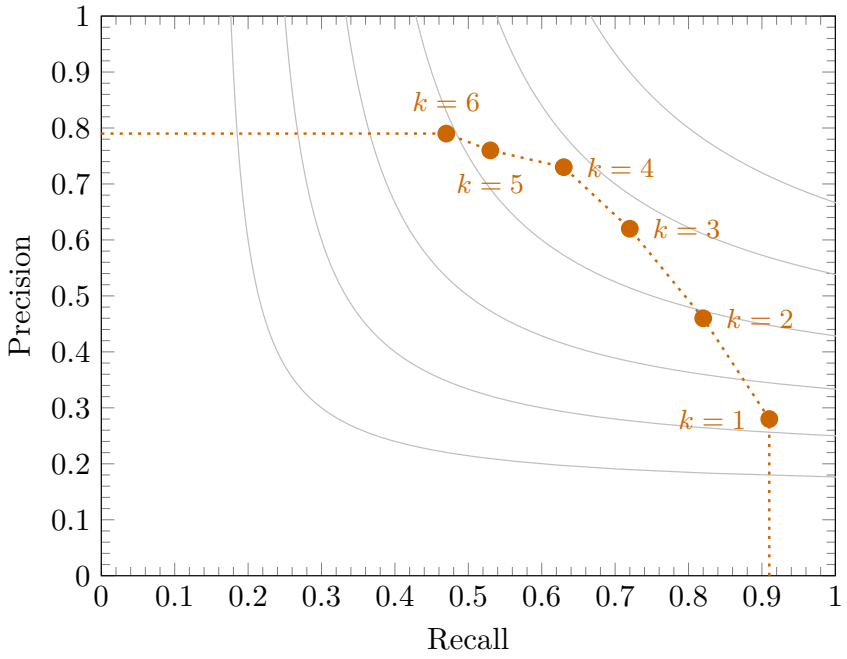


Figure 5.5: Precision – recall curve for the comparison between the p-NN graphs (for the pattern see figure 5.2) and the k-NN graphs. Markers represent evaluated configurations. Dotted lines are given to guide the eye. Isometric lines for F1-Score are shown in grey. Values taken from ref. [Neu24a].

independently by one FPGA, and the overlapping of the sectors ensures that no data is lost. The overlapping sectors must be merged in subsequent reconstruction steps that are not part of the graph-building stage.

To implement the online graph-building module, JSON databases are generated for each ϕ-sector of the CDC. Each database represents a formal detector that contains the positions of the wires and information about sensor features, as described in section 4.2. Sense wire features are composed of 1 bit for the binary hit classifier, 5 bit for the TDC, 4 bit for the ADC, and the Cartesian coordinates of the wires. The bitwidth of the sense wire features are define by the CDC frontend electronics. Edge features contain information about the wire distances between two adjacent verices. The resolution of the edge features can be arbitrarily chosen and is therefore considered a hyperparameter of the module implementation.

The sector database and a function describing the pattern, as illustrated in figure 5.2, are provided as input to the proposed toolchain, which is implemented in Python 3.10. An intermediate graph representation is generated as a JSON database, containing type defi-

nitions for all vertices, edges, and their respective features. In addition, features known at design-time, such as Cartesian coordinates, are rounded, quantised, and included in the intermediate graph representation. By generating databases for all 20 sectors, the smallest and largest sectors of the CDC are identified, providing lower and upper bounds on the problem size. The maximum number of edges in each sector is determined by the pattern from figure 5.2. The smallest sectors are located in superlayer two, containing 498 vertices and 2305 edges, while the largest sectors are located in superlayer six, containing 978 vertices and 4545 edges.

The generated graph database will be used in the following section to generate the architecture of the GNN hardware accelerator.

Table 5.1: Partitioning of the Belle II CDC into 20 overlapping sectors, grouped by CDC superlayer. The relative overhead quantifies the fraction of wires and edges that are duplicated across neighbouring sectors to ensure full track coverage at sector boundaries.

Sectors	Superlayer	Wires	Edges	Overhead Wires (%)	Overhead Edges (%)
0, 1	0	664	3293	3.01	2.61
2, 3	1	498	2305	4.01	3.73
4, 5	2	588	2725	3.40	3.16
6, 7	3	691	3201	2.89	2.69
8, 9	4	786	3649	2.54	2.36
10, 11	5	882	4097	2.40	2.10
12, 13	6	978	4545	2.04	1.89
14, 15, 16	7	730	3372	2.73	2.55
17, 18, 19	8	786	3649	2.54	2.36
Total		14 722	68 693	—	—

5.4 System Architecture

To map the Interaction Network onto the FPGA, a dataflow accelerator architecture is developed. This section describes the resulting architecture for the model introduced in section 5.1, the algorithmic performance of which is evaluated in ref. [49]. The approach builds on the architecture templates presented in chapter 4. Further details on the deployment flow are provided in section 5.5.

An overview of the dataflow accelerator is given in figure 5.6. The dataflow is oriented from left to right. Each block corresponds to a hierarchical module, specifically one PE, and performs a distinct operation within the original neural network. The network is partitioned into seven PEs, three FIFOs, and a final filter stage. PEs mapping scatter and aggregate operations are

shown in blue, PEs mapping trainable network layers in green, and auxiliary modules such as FIFO buffers in grey.

The system receives a parallel input vector containing all wires of the respective sector of the Belle II CDC, up to 978 wires. For each wire, a 1 bit HIT flag, a 4 bit ADC readout, and a 5 bit TDC readout are supplied. The input port is AXI-Stream compliant and uses a ready/valid handshake. The output is provided through a second AXI-Stream port of identical width. This allows the system to act as a hit-filtering module: the algorithm masks wires by clearing the corresponding HIT flag.

Four types of PEs are required to map the Interaction Network onto the dataflow architecture.

Static Scatter PEs use the static graph derived at design time (see section 5.3) to transform features from the vertex level to the edge level, $f : X \to E$ (①, ⑤). Two key operations are performed: first, static edge and vertex features for the respective wires and edges are loaded from read-only memory. Second, both dynamic and static features are multicast and concatenated to form the requested edge feature vectors. Depending on the network topology, a Scatter PE may receive additional dynamic features via a FIFO interface from previous layers (feed-forward connection). The output is forwarded to the succeeding Edge PE.

Edge Transform PEs map the inference pass of differentiable, learnable functions onto the FPGA (②, ⑥). These functions are typically simple MLPs represented by a sequence of dense layers. Each edge feature vector is processed by an identical MLP, and no intermediate result of one edge is required for the computation of another. This permits a parallelisation of the Edge PE stage across edges, limited only by the available hardware resources. The output of the Edge PEs is forwarded to a static aggregate PE.

Static Aggregate PEs implement the inverse transformation with respect to the static scatter PEs: $f : E \to X$ (③, ⑦). Edge feature vectors are reduced to vertex features through a permutation-invariant function, such as the summation or the maximum operator. Network designers may consider that a summation-based aggregation requires additional bits proportional to $\log_2(N)$, where N denotes the fan-in of the aggregation. The assignment of edges to their respective output vertex is derived from the design-time graph. The output of this PE is forwarded either to a Vertex PE or to the output filtering stage of the network.

Vertex Transform PEs map the inference pass of differentiable, learnable functions onto the FPGA (④). Their functionality is identical to that of the Edge PEs. They are distinguished

in the architecture diagram solely for clarity. Rather than applying a transformation to each edge, the transformation is applied to each vertex. As with the Edge PEs, the vertex-level transformations are data-independent across vertices, so this stage can likewise be extensively parallelised, limited only by available hardware resources. The degree of parallelism at this stage is a trade-off between inference latency and resource usage, as discussed further in section 5.5.

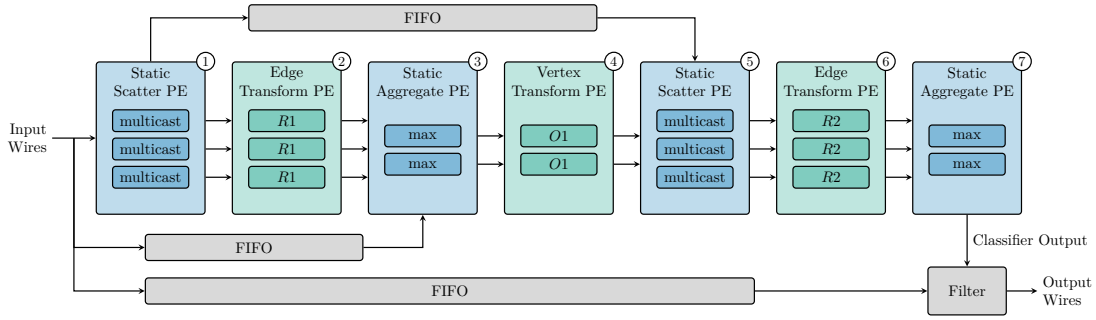


Figure 5.6: Architecture of the dataflow accelerator for the model given in figure 5.1. PEs mapping static scatter and aggregate operators are shown in blue, PEs mapping trainable network layers in green, and auxiliary modules such as FIFO buffers in grey. Adapted from ref. [May25].

Five of the seven PEs shown in figure 5.6 correspond directly to the layers of the Interaction Network introduced in section 5.1. The two PEs without a direct counterpart in the algorithmic description are the static scatter PEs ① and ⑤. As multicast is a trivial operation from an algorithmic perspective, it is only shown implicitly in the algorithm-level description.

In figure 5.6, the output of the final static aggregate PE (⑦) is a scalar value per wire. This scalar represents a prediction score for the corresponding wire on a signal track. Lower scores indicate a higher probability that the hit originates from crosstalk or beam background. Based on a run-time, user-defined threshold, the filter PE masks all wires whose prediction score lies below this threshold. In this manner, a hit-filtering operation is applied to all input wires.

A set of algorithm-independent design parameters exists which influence the resulting throughput directly and are thus discussed below: First, the Edge and Vertex PEs may be parallelised independently of one another, each by an individual parallelisation factor p_i, where i identifies the PE stage, for example ②, ④, or ⑥. Second, the scatter and aggregate PEs require that all input edges or vertices be available before their respective outputs are produced, since the order of read and write accesses depends on the static graph and is not

resolved at design time. The scatter and aggregate PEs are therefore realised as a double-buffered module. A decomposition into FIFOs is not feasible because individual features may be read multiple times depending on the fan-out of the static graph. Third, the FIFOs between successive PEs must be dimensioned such that no stalls occur during operation, as an undersized FIFOs otherwise limit the achievable throughput of the system. Given the parallelisation factors of all PEs, the throughput T of the accelerator is determined by the slowest stage and can be calculated at design time as

$$T = \frac{f_{\text{sys}}}{\max\left(\max_{i \in \mathcal{V}} \left\lceil \frac{|\mathcal{V}|}{p_i} \right\rceil, \max_{j \in \mathcal{E}} \left\lceil \frac{|\mathcal{E}|}{p_j} \right\rceil\right)}, \tag{5.3}$$

where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of Vertex and Edge PEs, respectively, $|\mathcal{V}|$ and $|\mathcal{E}|$ are the numbers of vertices and edges of the static graph, and f_{sys} is the clock frequency of the accelerator.

5.5 Deployment and System Design

Deploying the Interaction Network onto the FPGA requires translating the algorithm description of section 5.1 into a hardware implementation that satisfies the resource and timing constraints of the L1 Trigger at Belle II. The toolflow used for this translation is outlined in section 5.5.1. Two aspects of the deployment are examined in detail: Floorplanning optimisations are discussed in section 5.5.2. FIFO buffer optimisations are discussed in section 5.5.3.

5.5.1 Toolflow Methodology

The toolflow methodology described in this section addresses the translation of a given Interaction Network onto an FPGA using the DeePloy methodology (see chapter 4), which consists of nine design steps (Ⓐ – Ⓘ). An overview of this toolflow is shown in figure 5.7. Steps labelled as semi-automated allow changes to weights and data types without manual intervention, whereas changes to the network topology require some manual intervention. The toolflow takes four inputs:

Detector representation defines the set of all vertices. In the Belle II CDC, each wire represents one vertex. The detector representation also provides static features, such as the sense wire position and the distance to the interaction point.

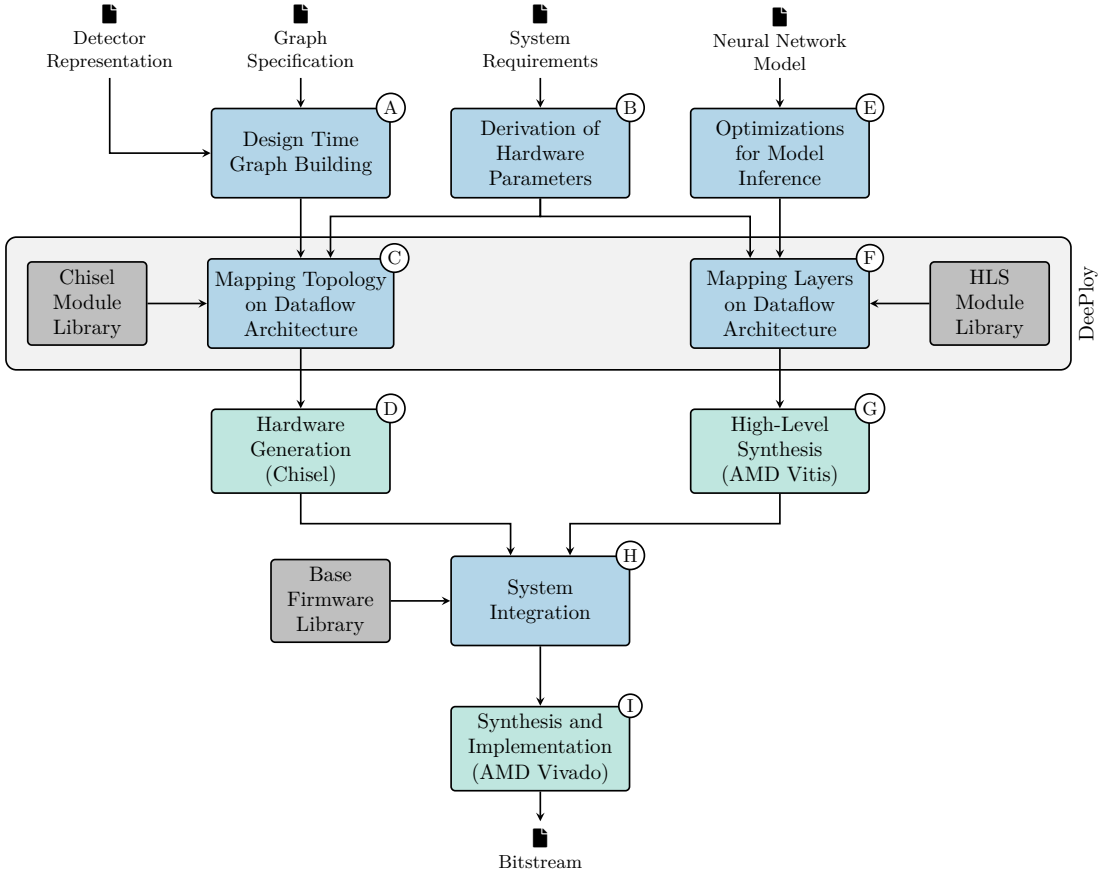


Figure 5.7: Overview of the toolflow methodology for the deployment of the Interaction Network onto FPGA. Design steps are labelled Ⓐ – Ⓘ and colour-coded by origin: steps shown in blue represent original contributions of this work, while steps shown in green employ existing electronic design automation tools. Supporting libraries are shown in grey.

Graph specification is either an ε-NN or a p-NN specification which can be applied to the detector. k-NN graph building is not supported due to the runtime dependence of neighbouring vertices.

System requirements are used to derive the appropriate parallelism such that the resulting hardware implementation reaches the target throughput R_{th}.

Neural network model contains the weights and quantisations of the trainable layers.

As an output, the flow produces a bitstream for the target platform. In the following, the nine design steps from Ⓐ to Ⓘ are outlined:

In Ⓐ, the design-time graph building step generates an intermediate graph representation

based on the provided detector representation and graph specification. This step is essential for mapping static PCNs layers, as it enables computation of the worst-case graph the neural network operates on.

In Ⓑ, hardware parameters are derived. For mapping the Interaction Network, the parallelisation factors P are most important. Based on the user-provided throughput requirement and equation (5.3), P is minimised for all layers such that the achieved throughput meets or exceeds R_{th}. Because the search space is small, the optimisation problem can be solved exhaustively.

In Ⓒ, the neural network topology is mapped onto the dataflow accelerator. This semi-automated step considers three sub-tasks: First, the top-level scaffold of the dataflow accelerator is generated. PEs are instantiated as black boxes at this stage; their generation is described in Ⓕ. Second, multicast, scatter and aggregate operations are mapped onto switchbox PEs from the Chisel library. The intermediate graph representation is used to generate the appropriate dataflow. Third, FIFOs are inserted for feedforward connections.

In Ⓓ, the top-level Chisel design is transformed into a SystemVerilog [166] design. For this study, Chisel 6.4.0 is used.

In Ⓔ, the model is optimised for inference. Because the network topology is handled separately, this step covers only the trainable layers that will be deployed using HLS templates. The optimisations performed are the decomposition of MLPs into a series of dense or linear layers and the fusion of requantisation operations into these layers.

In Ⓕ, the prepared networks are mapped onto the layer templates available in the DeePloy HLS module library. This semi-automated step loads weights and biases from the network definition and generates a Vitis project for each trainable MLP network (R1, O1, R2). As an output of this step, Vitis C++ is generated.

In Ⓖ, AMD Vitis HLS is used to generate SystemVerilog [166] from the Vitis C++ projects. For this study, AMD Vitis HLS 2024.2 is used.

In Ⓗ, the top-level scaffolding generated by Ⓓ and the PEs generated by Ⓖ are combined into a single RTL design. In addition, the generated design is integrated into the base firmware, which handles IO mapping and connections to external interfaces.

Finally, in Ⓘ, AMD Vivado 2024.2 is used for synthesis, placement, and routing of the design [167].

5.5.2 Floorplanning

As described in section 5.1, the complete Belle II CDC is partitioned across 20 FPGAs, with each platform implementing one CDC sector, as introduced in section 5.3. The sectorization of the CDC is dictated by the technical requirements of the input merger boards (see section 2.1.3) and therefore does not yield equally sized sectors. The largest sectors, located in superlayers 5 and 6, become the bottleneck for hardware implementation.

To mitigate routing congestion and aid timing closure across these sectors, the floorplan shown in figure 5.8 is used for all sectors in the subsequent evaluations in section 5.6. Each sector is partitioned into three sub-sectors, indicated in blue, with one sub-sector mapped to each of the three SLRs of the AMD Alveo V80. To eliminate inter-SLR communication, the input data at the sub-sector boundaries is duplicated across the neighbouring SLRs, increasing the effective overlap (as defined in section 5.3) by a factor of three. With this duplication, the input-wire overhead reaches approximately 6 % for the largest superlayer, compared to a baseline overhead of 2 % reported in table 5.1. Although this optimisation is specifically tailored to the three-SLR layout of the AMD Alveo V80, the same partitioning scheme can be transferred to the AMD UltraScale devices used on the UT 4 boards, which contain two or three SLRs depending on the device variant.

5.5.3 FIFO Buffer Optimizations

As shown in figure 5.6, the architecture contains three FIFOs in the network's feedforward connections. The high-throughput requirements of the dataflow accelerator necessitate a large parallelisation factor across all PEs. Consequently, the fan-in and fan-out of the resulting RTL netlist increase substantially compared to a sequential reference design.

During iterative design-space exploration, the placement of these feedforward FIFOs is identified as a major bottleneck in the place-and-route stage in AMD Vivado. The root cause lies in the ready/valid handshake of the FIFO modules: when a FIFO approaches capacity, it deasserts the `ready` signal at its input port to stall the upstream producer. This backpressure path is purely combinational and must fan out to all parallel producer lanes, which results in a high-fanout net with a long combinational path. For designs operating close to the maximum frequency or with high parallelisation factors, this path becomes timing-critical and impedes routability.

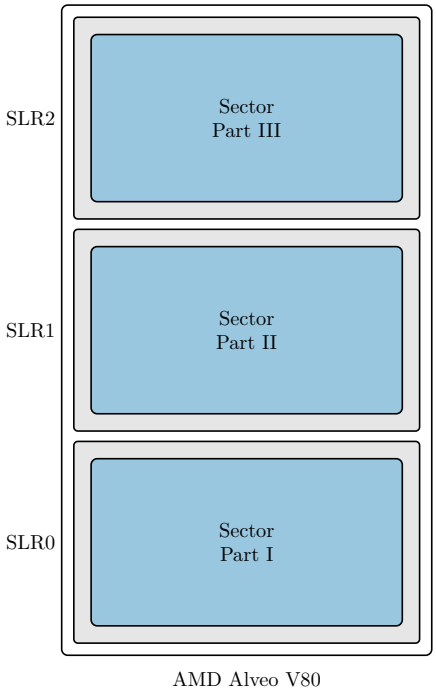


Figure 5.8: Floorplan constraints applied to the largest CDC sectors (superlayers 5 and 6) on multi-die FPGAs. The sector is partitioned into three sub-sectors (blue), each mapped to one SLR of the AMD Alveo V80. Boundary inputs are duplicated across neighbouring SLRs to eliminate inter-die routing on the critical paths.

The hardware accelerator developed in this thesis, however, is optimised for stall-free, high-throughput operation. Since the compute latency of all PEs is deterministic, a prerequisite for the hard real-time requirements of the trigger system, the required delay on each feedforward path can be derived analytically. The optimisation described in this subsection therefore replaces the generic FIFOs with shift registers of fixed length, where the depth of each shift register is determined by the analytically computed delay between the corresponding producer and consumer stages. Since the design is guaranteed to be stall-free by construction, the combinational backpressure path can be eliminated, substantially simplifying the netlist and relaxing timing constraints in the affected regions.

An alternative approach based on synchronous elastic circuits has been evaluated but did not yield the anticipated routability improvements [168]. The additional pipeline registers introduced by the elastic protocol offset the benefit of localised handshaking. Since the deterministic latency of the dataflow accelerator already eliminates the need for runtime stalling, the elasticity provided by such circuits offers no functional advantage sufficient to justify the resulting register overhead.

5.6 Performance Analysis

Experimental Setup

For evaluation of the system performance, the Interaction Network described in section 5.1 is deployed onto the AMD Alveo V80 evaluation platform. The deployment methodology described in section 5.5 is used to generate the dataflow accelerator architecture introduced in section 5.4. I have implemented the semi-automated design-automation flow in Python 3.11. The electronic design automation tools comprise AMD Vitis HLS 2024.2 [156] as the HLS toolchain, Chisel 6.4.0 [155] for hardware generation, and AMD Vivado 2024.2 [167] for synthesis, placement, and routing.

The GNN-HCM is implemented to ensure all design requirements are met. The static graphs from section 5.3 are used as a partitioning starting point to fulfil requirement 5.1. To achieve the system throughput requirements of requirement 5.3, the parallelisation of the design is chosen based on equation (5.3). A clock frequency of $f_{\text{sys}} = 127.22\,\text{MHz}$ is chosen because it is the base frequency of the Belle II L1 Trigger. In total, 20 designs are generated, one for each sector of the CDC.

Functional Verification

For validation, the generated hardware accelerator is tested in RTL simulation using functional verification methodology. RTL simulation is performed using ModelSim 2023.4 [169], and CocoTB is used to drive the interfaces in the testbench. For the tests, randomly generated input graphs representing the respective sector of the CDC are used as stimuli for the design under test. The result of the RTL simulation is checked against the golden model from Brevitas using PyTorch [147] and PyTorch Geometric [73]. My tests show bit-accurate agreement between the golden model and the device under test. In addition, the test cases are used to validate that the design runs stall-free for multiple inference periods.

Implementation Results

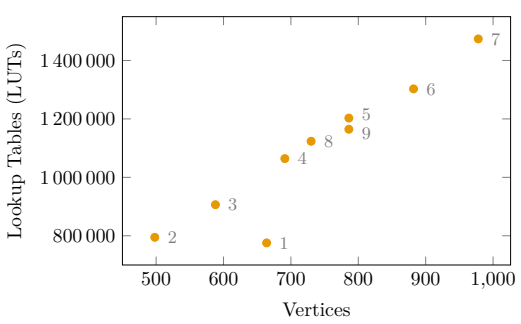
To evaluate that the GNN-HCM fulfils all requirements, the design is synthesised, placed, and routed out-of-context for the AMD Alveo V80 with f_{sys}. For synthesis, `PerformanceOptimized` is selected as a directive, retiming is enabled via `-retiming`,

and the minimum shift register size is set to `-shreg_min_size 128`. For implementation, the design is optimised with `ExploreWithRemap`, placed with `ExtraTimingOpt` ↪ , and further optimised and routed with `AggressiveExplore`. All timing constraints are verified as satisfied after routing for all designs at the specified design frequency.

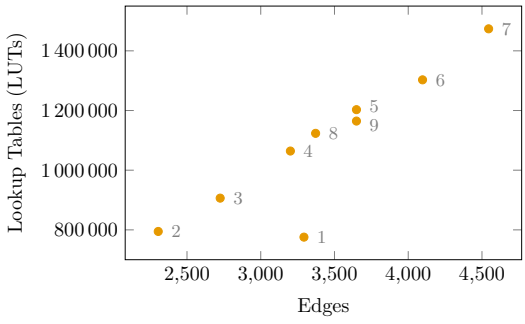
Resource Utilization Analysis

Based on the validated timing constraints and the cycle count derived from the RTL simulation, all 20 designs achieve a latency of 425 ns, thereby satisfying the 500 ns budget defined in requirement 5.2. The resource utilisation for all sectors is given in table 5.2. The largest sector is superlayer six, which requires 54.03% of all available Flip-Flop Registers (FFs), 57.24% of all available LUTs, and 44.64% of all CARRY units. The design fits on the AMD Alveo V80 with sufficient margin to accommodate the base firmware, which is not included in the out-of-context implementation. No DSPs are used, as the maximum bit width of arithmetic operations is limited to 5 bit by the model quantisation during training. The high utilisation of FFs is due to the increased minimum size of shift registers. This modification is required, as otherwise chained FFs would be mapped onto more compact shift register primitives on the FPGA. This optimisation, however, would be undesirable for this design, as additional FFs relax timing and congestion across the chip.

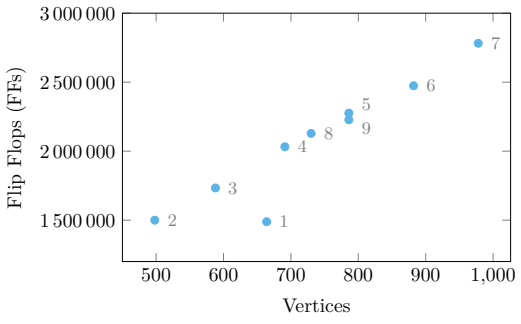
A different view of the design utilisation is given in figure 5.9. By plotting utilisation against the number of vertices and edges in each sector, a clear linear trend is evident: utilisation scales linearly with the number of vertices and edges across all types of system resources. One sector deviates from this pattern: superlayer zero. The reason lies in the different number of layers within this superlayer, resulting in a graph with a different topology.



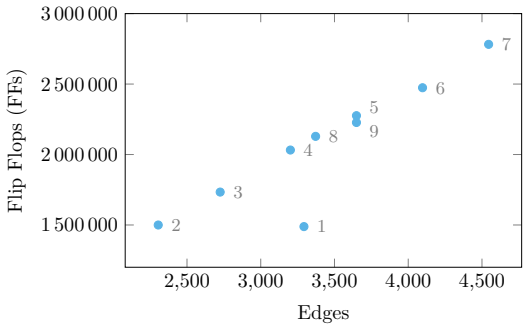
(a) LUTs over vertices.



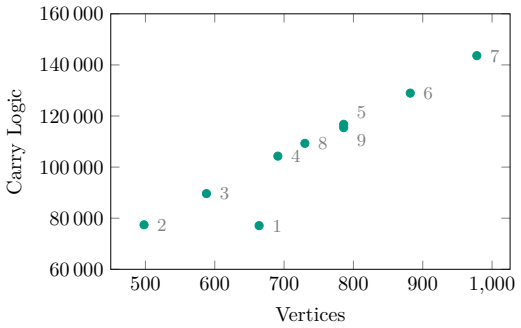
(b) LUTs over edges.



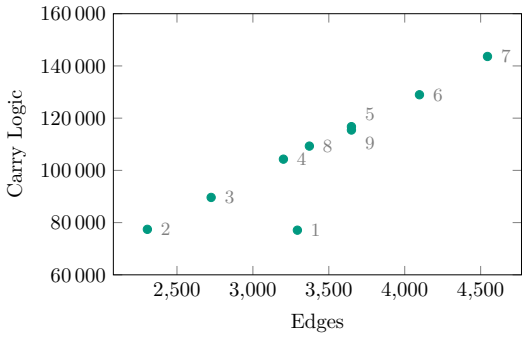
(c) FFs over vertices.



(d) FFs over edges.



(e) CARRY cells over vertices.



(f) CARRY cells over edges.

Figure 5.9: Resource utilization of the GNN-HCM implementations on the AMD Alveo V80 after out-of-context implementation, plotted against the number of vertices and edges in the graph. Grey numbers denote the superlayer of the respective sector shown.

Table 5.2: System resource utilization after out-of-context implementation on the AMD Alveo V80 evaluation board.

		Programmable Logic							
		FF		LUT		CARRY		DSP	
Sectors	Superlayer	abs.	%	abs.	%	abs.	%	abs.	%
0, 1	0	1 488 549	28.92	775 659	30.12	77 100	23.97	0	0.00
2, 3	1	1 499 811	29.13	794 694	30.87	77 424	24.06	0	0.00
4, 5	2	1 733 487	33.66	906 345	35.22	89 637	27.87	0	0.00
6, 7	3	2 031 837	39.48	1 064 187	41.34	104 298	32.40	0	0.00
8, 9	4	2 227 239	43.26	1 164 633	45.24	115 452	35.88	0	0.00
10, 11	5	2 473 956	48.06	1 302 810	50.61	128 961	40.08	0	0.00
12, 13	6	2 781 828	54.03	1 473 777	57.24	143 622	44.64	0	0.00
14, 15, 16	7	2 128 881	41.34	1 123 494	43.65	109 299	33.96	0	0.00
17, 18, 19	8	2 275 704	44.19	1 202 850	46.74	116 748	36.27	0	0.00

5.7 Discussion

In this chapter, I have presented the GNN-HCM case study for the Belle II CDC L1 Trigger, demonstrating the deployment of a static graph-based PCN, the Interaction Network, onto FPGA. First, I have analysed three graph building approaches for the Belle II CDC: k-NN, ε-NN, and p-NN. The ε-NN and p-NN algorithms I have presented achieve a $\mathcal{O}(1)$ time complexity and a $\mathcal{O}(|E|)$ space complexity, compared to a $\mathcal{O}(|V|)$ time complexity in approximate k-NN algorithms or a $\mathcal{O}(k|V|\log(|V|))$ complexity in the sequential case [79, Neu24a, 150]. For this case study, I have selected the p-NN graph building as specified in ref. [49] and partitioned the CDC into 20 sectors. Second, I have designed a dataflow accelerator architecture that maps the Interaction Network, including the graph-building step, onto an FPGA. To support this architecture, I have developed a semi-automated deployment flow that generates the per-sector accelerator from a common template, enabling the consistent deployment across all 20 sectors with their differing graph sizes. To demonstrate the feasibility of my approach, I have implemented the Interaction Network out of context on the AMD Versal V80 across all sectors.

My deployment approach enables the deployment of the Interaction Network for the CDC on all 20 sectors (requirement 5.1). My design achieves the required inference latency of 500 ns (requirement 5.2) and the throughput requirement of 32 million detector snapshots per second across all sectors (requirement 5.3). The largest sector, located in superlayer six, occupies 57.24 % of all LUTs on the AMD Alveo V80, leaving sufficient remaining resources

for the base design and IO.

However, it should be noted that the AMD Alveo V80 is one of the largest and latest FPGA platforms available. A direct application in the Belle II CDC L1 Trigger therefore remains infeasible in the near future, even after Long Shutdown 2 in 2030. While 20 UT 4 boards are expected to become available, the largest FPGA currently in use, the AMD Ultrascale XCVU190, provides roughly a factor of 2.5 fewer logic resources than the AMD Versal V80 on an older technology node. Closing this gap will require additional model compression beyond what I have applied in this work. Suitable candidates are LUT-based neural networks, such as NeuralLUT [170], which have recently been combined with high-granularity quantisation to enable smaller, faster model inference [171]. This direction is orthogonal to my deployment approach and can be integrated with comparatively little effort, as already demonstrated in a student thesis I have supervised [May25].

In summary, this chapter presents the first deployment of the Interaction Network for the Belle II experiment. Compared to the previous work on the Interaction Network in ref. [111], my approach achieves a 9.92× higher throughput and a 4.7× lower latency at a comparable graph size of up to 978 vertices and 4545 edges, versus 739 vertices and 1252 edges in the related work. A direct comparison is limited by differences in the target FPGA and the network configuration. The absence of publicly available source code further prevents a fully controlled benchmark. With this work, I provide the first end-to-end demonstration that the Interaction Network, including its graph building step, can be deployed within the latency and throughput constraints of the Belle II CDC L1 Trigger, and I establish a concrete reference design against which future GNN-based trigger implementations in Belle II can be evaluated.

Chapter 6

Real-Time Graph Neural Network Clustering at Belle II

The work presented in this chapter has been partly published in refs. [Neu26a, Neu26d].

The current Belle II ECL L1 Trigger is a multi-stage trigger implemented on FPGAs, responsible for identifying energy positions in real-time (see also section 2.1.5). It employs a clustering algorithm originally developed for the Belle experiment, which has proven effective under its original operating conditions. However, the system was designed with fixed logic and the limited resources of early FPGA generations in mind. Thus, the current system is limited in its scalability, clustering granularity, and adaptability to changing detector operating conditions. For instance, only a maximum of six trigger clusters can be processed due to FPGA resource limitations, duplicated trigger clusters are not filtered, two close-by photons cannot be resolved, and high-energy photon cluster position resolution is insufficient. Such limitations have a disproportionately large impact on events with only a few low-energy particles. The increase in detector hits originating from beam-induced backgrounds further reduces the performance of the current system. These limitations motivate the development of a more flexible and accurate approach. To address these challenges, the GNN-ETM has been developed and deployed as a real-time hardware trigger implemented on FPGA. GNN-ETM constitutes a hardware implementation of a PCN-based clustering algorithm within the Belle II L1 Trigger system, designed to achieve improved position resolution and photon separation at latencies compatible with the L1 Trigger constraints.

The remainder of this chapter is structured as follows. Section 6.1 introduces the underlying PCN model and derives the design requirements. Section 6.2 describes the realized trigger bits on the GNN-ETM. Section 6.3 describes the hardware architecture of the GNN-ETM. Section 6.4 details the deployment and optimization strategies. Section 6.5 covers the software infrastructure supporting the system. Section 6.6 discusses the validation methodology and results. Section 6.7 presents the performance evaluation. Section 6.8 describes the deployment and operation within the Belle II trigger system. Section 6.9 summarizes and discussed the findings.

6.1 Algorithm and Design Requirements

The development of the algorithm and the training of the neural network have been thoroughly studied in ref. [Neu26a]. In the following, the model is summarised to provide the necessary context for deriving the system requirements.

The model used for the GNN-ETM is a dynamic PCN, later on called CaloClusterNet, consisting of GravNet layers (see also section 2.2.3) and the condensation point selection algorithm (see also section 2.2.4). Figure 6.1 shows an illustration of the neural network model. As input, the model receives 576 trigger cells from the ECL detector, each with five features: time, energy, and the x, y, and z coordinates, defined as the mean of the crystal centers it contains. While there are 576 trigger cells in total, in most events, fewer than 32 trigger cells are hit at the same time, as shown in ref. [Neu26a]. Thus, the network is optimised for 32 trigger cells as input: Events with more than 32 trigger cells are truncated, whereas events with fewer than 32 trigger cells are zero-padded. The inference works as follows:

In a first step, trigger cell features are normalised, time and position to a range between -1 and 1, and the energy is normalised to a range of 0 to 1. In the second step, inputs are transformed through the dynamic PCN layers, namely the GravNet layer, into condensation point candidates. Each condensation point candidate contains the prediction of a potential cluster: an energy, a position and a background classifier. In the last step, the condensation point selection algorithm is applied to the condensation point candidates to identify the actual clusters and separate signal hits from background. Because the number of clusters is not known a priori, the network learns to derive the number of clusters through the object condensation algorithm during training [80]. Clusters predicted by the CaloClusterNet are used to calculate trigger bits for the Belle II ECL L1 Trigger (see also section 2.1.1).

To deploy the described algorithm, along with the calculation of dedicated trigger bits, into the Belle II L1 Trigger system, a dedicated FPGA-based hardware module is required. Because the system will fulfil the same role as the current ICN-ETM, it is placed in parallel in the trigger chain. Due to the limited number of physical connectors on the TMM, trigger cell data must be forwarded through the ICN-ETM FPGA module. The GNN-ETM trigger bits are forwarded to the GDL for the purpose of participating in the active trigger decision as well as monitoring. For debugging and monitoring purposes, two interfaces exist: First, a low-speed interface for connecting GNN-ETM to a general-purpose server via VME protocol. Second, a high-speed interface for connecting GNN-ETM to the Belle II readout system

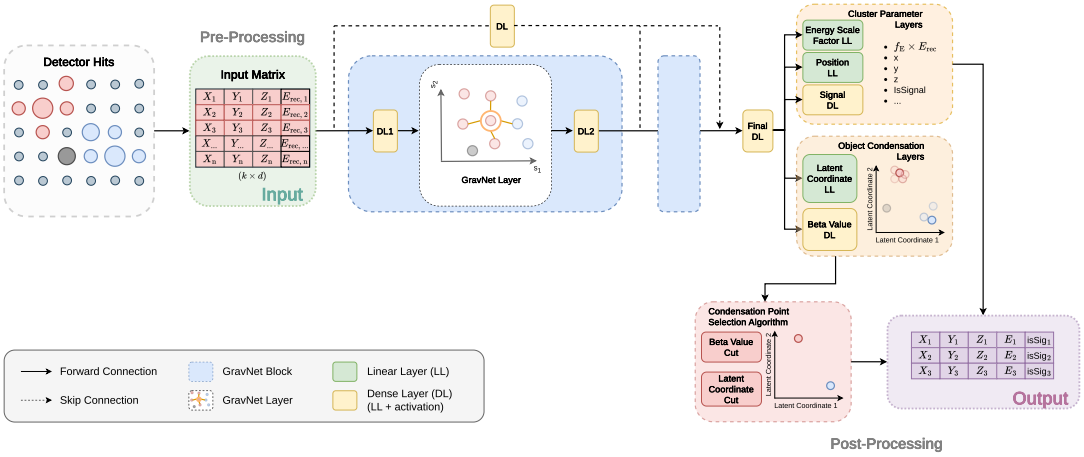


Figure 6.1: Illustration of the CaloClusterNet model from ref. [Neu26a].

via the Belle2Link protocol. A simplified schematic of the full integration of the GNN-ETM module is depicted in figure 6.2.

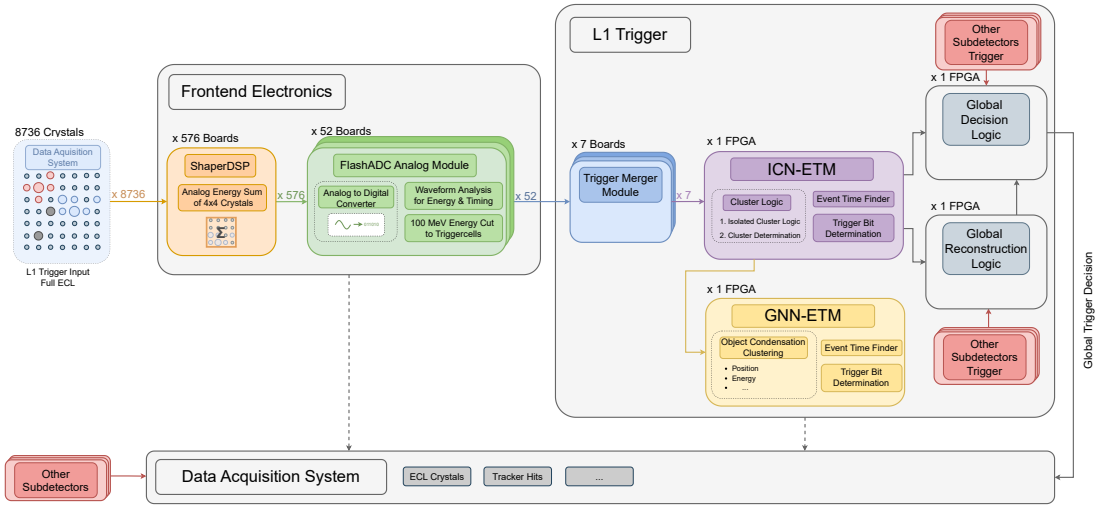


Figure 6.2: Overview of the current adapted ECL L1 Trigger chain at the Belle II experiment. VME and Belle2Link interfaces are omitted for clarity. Adapted from ref. [Neu26a].

Based on the algorithm and its position within the trigger hierarchy, four system requirements are derived for the GNN-ETM.

Requirement 6.1. The GNN-ETM shall deploy the full CaloClusterNet inference pipeline, including preprocessing and postprocessing, onto a single UT 4 platform.

Requirement 6.2. Trigger bits shall be available on the GDL before any buffer of the DAQ

system reaches its limit. A critical path latency of $R_{\mathrm{L}} = 5.0\,\mu\mathrm{s}$ is imposed by the Silicon Vertex Detector (SVD). The latency budget on the GNN-ETM is much smaller.

Requirement 6.3. The GNN-ETM shall sustain the full input rate of $R_{\mathrm{th}} = 8$ million detector snapshots per second. Each snapshot contains the values of all 576 trigger cells. Under current operating conditions, I observe no more than 32 TCs are active per snapshot.

Requirement 6.4. The GNN-ETM should operate with 100% uptime, as any disruption to the L1 Trigger system halts the entire experiment for the duration of the fault.

6.2 Trigger Bits

Trigger bits are boolean flags generated by modules in the L1 Trigger system. Their purpose is to classify trigger windows based on reconstructed physical quantities in the detector. As soon as at least one trigger bit is true, the event will be forwarded to the high-level trigger. If all trigger bits are false, the event will be discarded. The GDL aggregates trigger bits from all submodules to derive the global trigger decision. This signals the DAQ system to read out the detector snapshot and forward the event to the High-Level Trigger System (HLT) for further processing. For more background information, the interested reader may refer to section 2.1.1.

The GNN-ETM calculates 25 trigger bits in the postprocessing stage (see section 6.3.3), which are forwarded to the GDL via an optical interface. In addition, one trigger bit can be selected via the configuration register to be sent over a twisted-pair cable at lower latency. Trigger bits on the GNN-ETM are grouped into three categories: cluster-counting trigger bits, high-energy trigger bits, and low-multiplicity trigger bits.

For the following, the output of the CaloClusterNet algorithm is described as a set of clusters $\mathcal{C}$ with cardinality $|\mathcal{C}| = n_{\mathrm{TCs}}^{\mathrm{max}}$. Each cluster $c \in \mathcal{C}$ has an associated energy $E(c) > 0$, a condensation point flag $\mathrm{cpt}(c) \in \{0,1\}$, a signal flag $\mathrm{sig}(c) \in \{0,1\}$, and a TC theta index $\theta_{\mathrm{id}}(c) \in \{1,...,17\}$. The condensation point flag generated by the object condensation algorithm and indicates that this TC is predicted as cluster. The signal flag generated by the neural network is a binary classifier which predicts if a cluster is considered background or signal. The TC theta index is defined as a binned θ angle, where $\theta_{\mathrm{id}}(c) \in \{1,...,3\}$ resolves to TCs in the forward endcap and $\theta_{\mathrm{id}}(c) \in \{16,17\}$ resolves to the backward endcap. Generally, the definitions are similar or identical to existing trigger bits on the ICN-ETM. However, the Laborsystem is used for all trigger bits on GNN-ETM.

Cluster counting trigger bits assert that the number of clusters satisfying a set of quality criteria meets or exceeds a required count c_{req}. Without the signal classifier, only the condensation point flag is required:

$$\left|\left\{c \in \mathcal{C} : \text{cpt}(c) \wedge 1 < \theta_{\text{id}}(c) < 16 \wedge E(c) > E_{\text{thres}}\right\}\right| \geq c_{\text{req}}, \tag{6.1}$$

and with the signal classifier:

$$\left|\left\{c \in \mathcal{C} : \text{sig}(c) \wedge 1 < \theta_{\text{id}}(c) < 16 \wedge E(c) > E_{\text{thres}}\right\}\right| \geq c_{\text{req}}. \tag{6.2}$$

The GNN-ETM evaluates all permutations of equations (6.1) and (6.2) for the required cluster counts $c_{\text{req}} \in \{2,3,4\}$ and the energy thresholds $E_{\text{thres}} \in \{100\,\text{MeV}, 125\,\text{MeV}, 150\,\text{MeV}\}$, yielding 18 trigger bits in total.

High-energy trigger bits assert that the total energy deposited by clusters in the inner ECL region exceeds a fixed threshold. Without the signal classifier:

$$\sum_{\substack{c \in \mathcal{C} \\ \text{cpt}(c) \wedge 1 < \theta_{\text{id}}(c) < 16}} E(c) \geq E_{\text{thres}}, \tag{6.3}$$

and with the signal classifier:

$$\sum_{\substack{c \in \mathcal{C} \\ \text{sig}(c) \wedge 1 < \theta_{\text{id}}(c) < 16}} E(c) \geq E_{\text{thres}}, \tag{6.4}$$

where $E_{\text{thres}} \in \{1\,\text{GeV}, 1.5\,\text{GeV}\}$. The GNN-ETM evaluates all permutations of equations (6.3) and (6.4) for both energy thresholds, yielding four high-energy trigger bits in total.

Low-multiplicity trigger bits target single-cluster signatures that do not result from $e^+e^- \to \Upsilon(4S)$ decay. On GNN-ETM, three low-multiplicity trigger bits are implemented. The naming originates from the ICN-ETM trigger bits, of which a subset is implemented on the GNN-ETM. The trigger bit lml_1 is described by:

$$\text{lml}_1 = \left|\left\{c \in \mathcal{C} : \text{sig}(c) \wedge E(c) > 2\,\text{GeV} \wedge 3 < \theta_{\text{id}}(c) < 15\right\}\right| \geq 1. \tag{6.5}$$

The trigger bit lml_6 is described by:

$$\mathrm{lml}_6^1 = \left|\{c \in \mathcal{C} : \mathrm{sig}(c) \wedge E(c) > 1\,\mathrm{GeV} \wedge 3 < \theta_{\mathrm{id}}(c) < 16\}\right| = 1, \tag{6.6}$$

$$\mathrm{lml}_6^2 = \left|\{c \in \mathcal{C} : \mathrm{sig}(c) \wedge E(c) > 0.3\,\mathrm{GeV}\}\right| = 1, \tag{6.7}$$

$$\mathrm{lml}_6 = \mathrm{lml}_6^1 \wedge \mathrm{lml}_6^2 \tag{6.8}$$

The trigger bit lml_7 is described by:

$$\mathrm{lml}_7^1 = \left|\{c \in \mathcal{C} : \mathrm{sig}(c) \wedge E(c) > 1\,\mathrm{GeV} \wedge \theta_{\mathrm{id}}(c) \in \{2,3,16\}\}\right| = 1, \tag{6.9}$$

$$\mathrm{lml}_7^2 = \left|\{c \in \mathcal{C} : \mathrm{sig}(c) \wedge E(c) > 0.3\,\mathrm{GeV}\}\right| = 1, \tag{6.10}$$

$$\mathrm{lml}_7 = \mathrm{lml}_7^1 \wedge \mathrm{lml}_7^2 \tag{6.11}$$

Together, the 18 cluster counting, four high-energy, and three low-multiplicity trigger bits sum to the 25 trigger bits forwarded to the GDL. A comprehensive list of all trigger bits is given in appendix B.

6.3 System Architecture

To support online inference of the real-time algorithm, a system architecture is developed for the UT 4. The standardised hardware board features three types of interfaces: First, for low-latency, high-bandwidth data transmission, high-speed gigabit transceivers are used on the FPGA. To minimise the bit-error rate, optical fibres are used for these high-speed links between successive hardware boards on the critical path of the L1 Trigger system. Inside the FPGA, interfaces to the transceivers are exposed as AXI-Stream interfaces [146]. Second, for slow control and monitoring, a VME interface is employed [46]. GNN-ETM configuration and status registers are exposed via VME bus to a host server. On the host server, a hardware abstraction Application Programming Interface (API) interacts directly with the VME kernel driver on the Linux operating system and exposes the status to the NSM 2 slow control system at Belle II. Third, for continuous monitoring, the Belle2Link physical layer protocol [37, 38] is used, which enables readout of data from the system in a manner synchronous with other components of the L1 Trigger system.

A hierarchical overview of the GNN-ETM architecture is shown in figure 6.3. All components and interfaces provided by the base firmware of the UT 4 are shown in grey. In total, six hardware modules are developed to facilitate the deployment of the CaloClusterNet algorithm

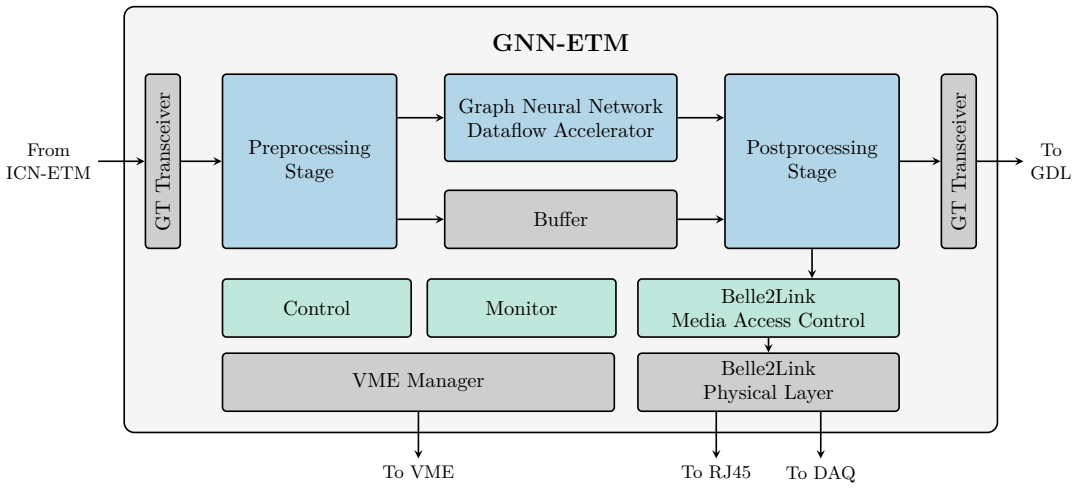


Figure 6.3: Overview of the GNN-ETM system architecture. Existing components of the base firmware are grey. Modules introduced in this work are coloured. Modules on the critical path of the trigger system are blue. The remaining modules are green. Adapted from ref. [Neu26a].

onto the FPGA. In figure 6.3, three out of six modules are on the critical path for the L1 Trigger latency:

The Preprocessing Stage unpacks data received from the ICN-ETM, checks their integrity, and performs a series of transformation steps on the TC data. The three key transformation steps are summarised as follows: First, the dense representation of 576 TCs is compressed into a sparse row format with up to 32 entries. Second, a timing calibration is performed selecting the most energetic TC in the trigger window (250 ns) as reference time zero. Third, depending on the deployed model, an embedding for the neural network input features is extracted from static LUTs. A more detailed description of this module is given in section 6.3.1.

The Graph Neural Network Dataflow Accelerator is composed of PEs connected by FIFO buffers. The dataflow architecture is semi-automatically generated based on the DeePloy methodology (see also chapter 4). This module is described in detail in section 6.3.2.

The Postprocessing Stage aggregates data from the two previous stages, the preprocessing stage and the dataflow accelerator. This module implements two important functionalities. First, output clusters are matched to their respective TC identifier. Second, binary trigger bits are calculated based on the predicted clusters and configuration parameters. The output

signals of this module are forwarded to the GDL for a final trigger decision. This module is described in detail in section 6.3.3.

In figure 6.3, three additional modules are not on the critical path:

The Belle2Link Media Access Control module performs data-level synchronisation and builds dynamically sized Belle2Link packets. Its purpose is the aggregation of all intermediate results on GNN-ETM, belonging to the same event. Belle2Link packets contain intermediate results after the preprocessing stage, the GNN clusters, and the calculated trigger bits. This module is described in detail in section 6.3.4.

The Control module receives data from the VME bus and sets internal configuration registers accordingly. Functionalities include a full software reset, parameters for the Condensation Point Selection algorithm, and time-offset configuration for the Belle2Link interface. The register map is described in appendix C.

The Monitoring module aggregates internal signals from GNN-ETM and makes them available for readout via the VME bus. Functionalities include clock counters, trigger bit monitors, diagnostics for physical connectors, and status registers. The corresponding register map is described in appendix C.

6.3.1 Preprocessing Stage

The preprocessing stage of the GNN-ETM consists of seven central modules shown in figure 6.4. It is implemented as a parametrisable hardware generator, and the modules are described in detail below:

The Unpacker module extracts the serialised data format which is received from the ICN-ETM. As an output, the unpacker module provides parallel FIFO buffers for further processing. For error detection, the continuity of the global clock counter provided by ICN-ETM is checked, and, in the event of a mismatch, a status bit is forwarded to the monitoring module. Details on the data format are given in appendix B.

The Address Generation module calculates the address of each incoming TC based on its position in the packet. Addresses are only assigned to active TCs. This step is necessary for the subsequent sparsity compression.

The Trigger Window module retains all TCs from previous data windows to build trigger windows (see also section 2.1.1). For the ECL, a data window is defined as a detector snapshot containing all $n_{TCs} = 576$ at once. Currently, data windows are 125 ns long, and are extended

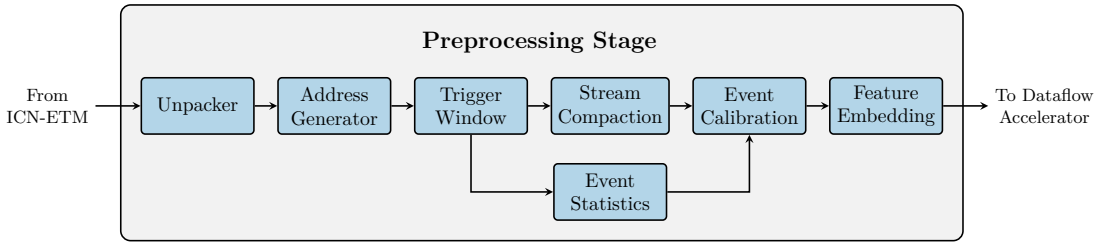


Figure 6.4: System overview of the preprocessing stage. Adapted from ref. [Neu26a].

to 250 ns long trigger windows. Effectively, two consecutive data windows are merged into a single trigger window. The FAM guarantees that no dublicate TCs are sent in a 250 ns time-frame, eliminating the need for handling these duplications.

The Stream Compaction module compresses the input matrix containing $n_{TCs} = 576$ rows into a smaller matrix with a maximum of $n_{TCs}^{max} = 32$ rows. For this module, the stream compaction approach described in section 4.6 is used.

The Event Statistics module receives all TCs from the trigger window module. It calculates characteristics per trigger window, such as the total number of active TCs and the highest energetic TC. The Event Calibration module subsequently uses this information. In addition, this information is essential for building dynamically sized Belle2Link packets later on and for realigning the intermediate datastreams of the firmware.

The Event Calibration module performs fast online timing calibration. Calibrated TC timing in each trigger window is computed relative to the timing of the most energetic TC in that trigger window. The effective timing offset per TC is now specified as a value in −250 ns to 250 ns.

The Feature Embedding module is the last step of the preprocessing stage. A LUT-based transformation is performed, effectively mapping TC identifier, energy, and calibrated timing information into a data format suitable for the CaloClusterNet model. The specific LUTs are typically dependent on the deployed neural network model. In the current firmware, the energy is divided by 8 to scale most values to the range 0 to 1. Timing and TC location (x, y, z) are normalised to the range -1 to 1. Values in the LUTs are stored in 16 bit fixed-point precision.

The preprocessing stage is implemented as a parametrisable hardware generator in the Chisel design language [155] and packaged as an IP core with FuseSoC [172, 173]. An overview of all design parameters for the preprocessing stage is shown in table 6.1. The values of the current

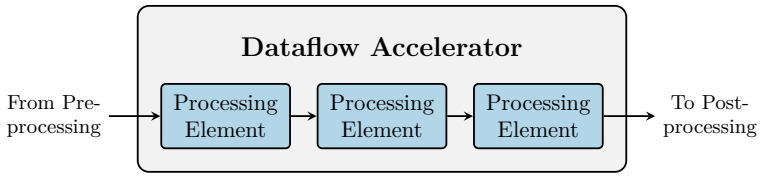


Figure 6.5: Typical system overview of the GNN dataflow accelerator. The specific versions of the CaloClusterNet deployed on the GNN-ETM are described in section 6.4.

implementation are given in the respective rows. The parameterisation allows a straightforward upgrade for future systems, when more than the current maximum of 32 active TCs are to be supported. To support, for example, 64 active TCs at the same data rate, the number of output channels must be increased from two to four.

Table 6.1: Adjustable parameters for the preprocessing stage and their current values.

Parameter	Value	Description
Timing Width	7 bit	Bitwidth of TC input timing feature.
Energy Width	12 bit	Bitwidth of TC input energy feature.
Address Width	10 bit	Bitwidth of TC address.
Input Channels	36	Number of parallel channels per data window.
Input Depth	32	Max. number of TCs per channel per data window.
Output Channels	2	Number of parallel channels per trigger window.
Output Depth	16	Max. number of TCs per channel per trigger window.
Trigger Windows	2	Number of data windows aggregated into a trigger window.

6.3.2 Graph Neural Network Dataflow Accelerator

The GNN Dataflow Accelerator consists of PEs connected by FIFO buffers, arranged in a left-to-right dataflow topology. Each PE is a free-running pipelined hardware module that processes inputs as soon as they are available. A typical system overview of the accelerator is shown in figure 6.5.

The control flow of the dataflow accelerator is governed by two model-independent parameters, listed in table 6.2. The number of TCs processed per cycle p and the total number of TCs per trigger window $n_{\mathrm{TCs}}^{\mathrm{max}}$ jointly determine the throughput T in trigger windows per second, given by equation (6.12). As the design employs stall-free pipelines, T is directly determined by the system clock frequency f_{sys}:

$$T = \left\lceil \frac{n_{\text{TCs}}^{\max}}{p} \right\rceil^{-1} f_{\text{sys}} \tag{6.12}$$

In contrast, the latency is model-dependent and is characterised after all HLS and hardware generation steps are complete, as described in section 6.4.

Table 6.2: Model-independent, adjustable parameters of the GNN dataflow accelerator and their currently used value in GNN-ETM.

Parameter	Value	Description
p	2	Number of TCs processed per cycle.
$n_{\text{TCs}}^{\max}$	32	Total number of TCs per trigger window.

The accelerator performs a single batch inference per trigger window. For each inference, it receives up to 32 input features from the preprocessing stage. If fewer than 32 TCs are present, the remaining inputs are zero-padded for alignment, and the corresponding outputs are zero-padded identically.

The dataflow accelerator provides three FIFO output interfaces. First, the primary inference output provides 32 cluster candidates, each described by five 16 bit features: a signal classifier, an energy prediction, and the position prediction (x, y, z). All 32 candidates are assembled in parallel into a single memory element of 2560 bit. Second, the CPS PE produces a 32 bit bitmap indicating, for each candidate, whether it should be forwarded to the postprocessing stage as a valid cluster. By decoupling this flag from the primary output, the postprocessing stage can speculatively compute metrics on all candidates in parallel, reducing overall latency. Third, a monitoring interface exposes all ten per-candidate features: the signal classifier, energy prediction, three-dimensional position prediction, three-dimensional latent space representation, β value, and cluster classifier. As this interface incurs higher latency, it is intended for use only on non-critical timing paths, such as the Belle2Link interface.

The thresholds for the β and isolation cuts used by the CPS PE are programmable via unsynchronised ports. Because these thresholds influence the model's algorithmic performance, they should be adjusted only between data-taking runs, when the L1 Trigger is idle and not processing any data. Internally, these ports connect to the control register described in appendix C.

During deployment, neural network layers are mapped to one or more PEs in the accelerator. As multiple design iterations of CaloClusterNet are deployed on the GNN-ETM, and all

adhere to the same interfaces described above, the deployment of each version is described separately in section 6.4.

6.3.3 Postprocessing Stage

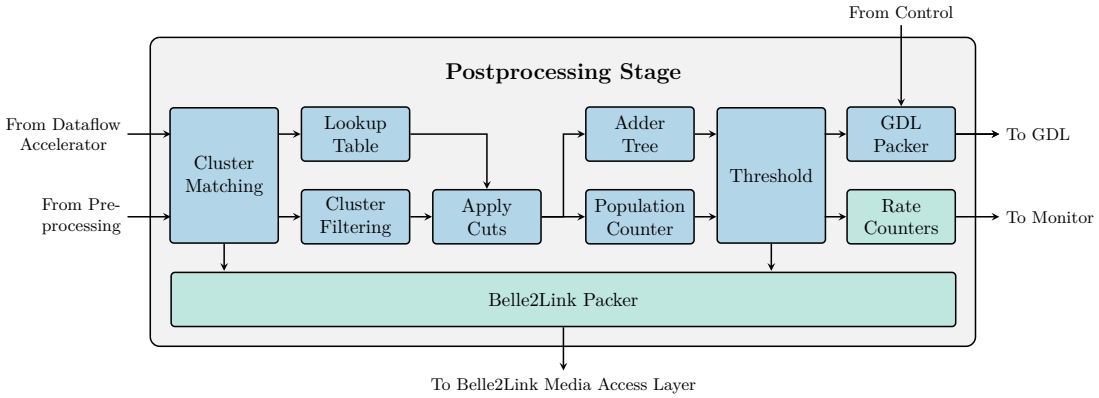


Figure 6.6: System overview of the postprocessing stage. Taken from ref. [Neu26d].

The postprocessing stage consists of ten functional modules, of which eight are on the critical timing path. A detailed overview is shown in figure 6.6. The dataflow in the figure is oriented from left to right and from top to bottom. It is implemented as a parametrisable hardware generator, and the modules are described in detail below:

The Cluster Matching module synchronizes the datastreams from the GNN dataflow accelerator (see section 6.3.2) and the preprocessing stage (see section 6.3.1). As the trigger cells are available much earlier than the predicted cluster information, their values are stored in FIFO buffers. Furthermore, predicted clusters from the GNN dataflow accelerator are matched to TCs from the same trigger window. Due to the strictly in-order processing of all clusters, this operation reduces to a simple concatenation of synchronised features.

The Lookup Table is a generic module for retrieving static information based on the TC identifier. Similarly to the preprocessing stage, where the location of a TC is retrieved, the module is used here to retrieve the appropriate θ_{id} for each cluster, which is required by the successive module.

The Cluster Filtering is performed in parallel to the lookup of the respective features. Since the dataflow accelerator produces results for all 32 zero-padded clusters, the CPS mask must

be applied to the output feature vectors. All inactive clusters are set to zero and thus ignored in subsequent pipeline steps.

The Apply Cuts module applies cuts for all respective trigger bits. An overview of the implemented trigger bits is given in section 6.4. In practice, this module implements a series of comparisons which filter the output. For example, a 100 MeV energy threshold is applied, setting all clusters below this value to zero.

The Adder Tree module is used for all sum-based trigger bits. It is implemented as a five-cycle pipelined structure and is used for the energy sum trigger bits. The number of pipeline cycles required scales with $\mathcal{O}(\log_2(n_{\mathrm{TCs}}^{\mathrm{max}}))$.

The Population Counter module is used for all cluster counting trigger bits. It computes a popcount over the active cluster flags remaining after the cuts applied by the preceding module.

The Threshold module is the final module for the calculation of trigger bits. It implements the Boolean equations which define the final trigger bits to be sent to the GDL. Its output is also used for monitoring the trigger rates on GNN-ETM via the slow control interface. **The GDL Packer** module prepares a 128 bit packet to be transmitted to the GDL via a gigabit transceiver interface. The full specification of the packet is described in appendix B. For reduced-latency operation, GNN-ETM also supports transmitting a copy of a trigger bit via a twisted-pair (LEMO) cable, which serves as an alternative path to the GDL. Using this cable substantially reduces signal latency between GNN-ETM and GDL. However, only one trigger bit can be transmitted this way.

The Rate Counter module implements a 32 bit counter for every trigger bit. This allows the slow control to read the accumulated trigger rates from GNN-ETM during operation. 32 bit are choosen for storage, as this size allows a continous counting of the trigger rates for 8 h at approx. 150 kHz.

The postprocessing stage is implemented as a parametrisable hardware generator in the Chisel design language [155] and packaged as an IP core with FuseSoC [172, 173]. An overview of all design parameters for the postprocessing stage is shown in table 6.3, along with the values of the current implementation.

Table 6.3: Adjustable parameters for the postprocessing stage and their current values.

Parameter	Value	Description
p	2	Number of TCs processed per cycle.
$n_{\text{TCs}}^{\text{max}}$	32	Total number of TCs per trigger window.
Features per TC	5	Number of features per TC from preprocessing.
Features per Cluster	5	Number of features per cluster on GDL output.
Features per Monitor	9	Number of features per cluster on Belle2Link monitor.
Feature Width	16 bit	Bitwidth per feature.
Energy Scale	2^{-11}	Significance of the last bit for the energy prediction.

6.3.4 Belle2Link Media Access Control

The Belle2Link Media Access Control module consists of six functional modules. Because this module is used for debugging purposes only, it is not on the critical path of the L1 Trigger system. The module is implemented as a parametrisable hardware generator in the Chisel design language [155] and packaged as an IP core with FuseSoC [172, 173]. The dataflow is shown in figure 6.7, proceeding from left to right. The module accepts AXI-Stream interfaces as inputs and generates a Belle2Link Physical Layer interface [37, 38] as output. It is derived from the original ICN-ETM design [56]. It has been extended to a more generalised form that supports an arbitrary number of AXI-Stream interfaces and a wider range of configuration parameters, as described below. The module performs data-level synchronisation across multiple AXI-Stream channels and assembles dynamically sized Belle2Link packets. The specific layout of the Belle2Link packets is described in more detail in appendix B. The module is composed of six submodules:

The Channel Alignment module performs data-level synchronisation of incoming AXI-Stream signals. On the GNN-ETM, the system is monitored between pipeline stages, i.e., the preprocessing stage, the dataflow accelerator, and the postprocessing stage. The channel alignment module ensures that the resulting Belle2Link packets contain monitoring data belonging to a single trigger window. Synchronisation is based on the arrival of the first `valid` signal: as soon as the last channel asserts `valid`, the module produces a synchronised output stream.

The Channel Delay and Trigger Delay modules together implement a two-stage alignment of the data path with respect to the incoming trigger signal. Because the propagation delay between the trigger signal at the GDL and its arrival at the GNN-ETM is unknown, an alignment procedure is required.

The Channel Delay module implements a programmable delay of the data path. By adjusting this delay, the offset between the incoming trigger signal and the data signal can be compensated, enabling the readout of trigger windows from preceding time steps that have already traversed the full pipeline. The granularity of this module is one data window (125 ns). The delay value is programmable via the GNN-ETM register map.

The Trigger Delay module implements a programmable delay of the trigger signal itself. Unlike the Channel Delay module, this module enables fine-grained alignment by artificially delaying the trigger signal at its input, effectively stepping forward in time by a resolution of a single f_{sys} cycle. Because the delay is applied only to the single-bit trigger signal, this module's hardware utilisation is significantly lower than that of the Channel Delay module. The delay value is programmable via the GNN-ETM register map.

The DAQ Buffer modules store trigger window data as part of the data acquisition logic. All channels are combined into a single packet in a dynamic format, in which each field is present only if the corresponding channel transmits data, as indicated by its `valid` signal. Empty trigger windows are skipped to reduce the load on the DAQ system.

The Dispatcher controls the state machine of all DAQ buffer modules. Incoming trigger signals are distributed in a round-robin fashion across the available buffers. Dispatching to multiple DAQ buffers in parallel allows multiple events to be recorded concurrently, even if the transmission of a preceding packet via Belle2Link has not yet completed.

The Arbiter module is a round-robin arbiter that manages access to the Belle2Link physical layer interface.

In addition to the basic functionality, the module exposes several monitoring registers. Specifically, the currently dispatched DAQ buffers are accessible via the register map. A safety mechanism also records whether a packet had to be dropped because all buffers were full. All monitoring registers are described further in section 6.5.

An overview of all design parameters for the module is given in table 6.4, along with the values used in the current implementation.

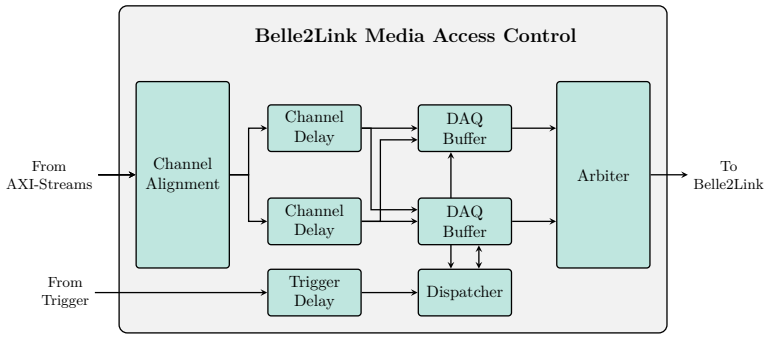


Figure 6.7: Overview of the Belle2Link subsystem. Adapted from ref. [Neu26a].

Table 6.4: Adjustable parameters of the Belle2Link Media Access Control module and the values used in the current implementation.

Parameter	Value	Description
Max Data Delay	16	Maximum delay for data signals, in trigger windows.
Num Data Windows	3	Number of trigger windows per Belle2Link packet.
Num Data Queues	12	Supported number of parallel active requests.
Max Trigger Delay	512	Maximum delay for the trigger signal, in f_{sys} cycles.

6.4 Deployment and System Design

Deploying the CaloClusterNet onto the GNN-ETM requires translating the algorithm description of section 6.1 into a hardware implementation that satisfies the resource and timing constraints of the L1 Trigger at Belle II. The toolflow used for this translation is outlined in section 6.4.1. Three aspects of the deployment are examined in detail: sparsity compression in section 6.4.2, and the mapping of the CaloClusterNet onto the dataflow accelerator in section 6.4.3. Finally, the physical optimisations applied to satisfy these constraints are described in section 6.4.4.

6.4.1 Toolflow Methodology

Deploying GNNs onto FPGAs for low-latency, high-throughput applications in the L1 Trigger and DAQ systems of high-energy particle detectors presents particular challenges. Hard real-time deadlines, deterministic behaviour, and strict reliability requirements necessitate careful co-adaptation of the network architecture and the deployment methodology to the target platform. As a key differentiation from existing flows, an end-to-end toolflow adapted specifically for the GNN-ETM is presented, integrating the Chisel-based hardware

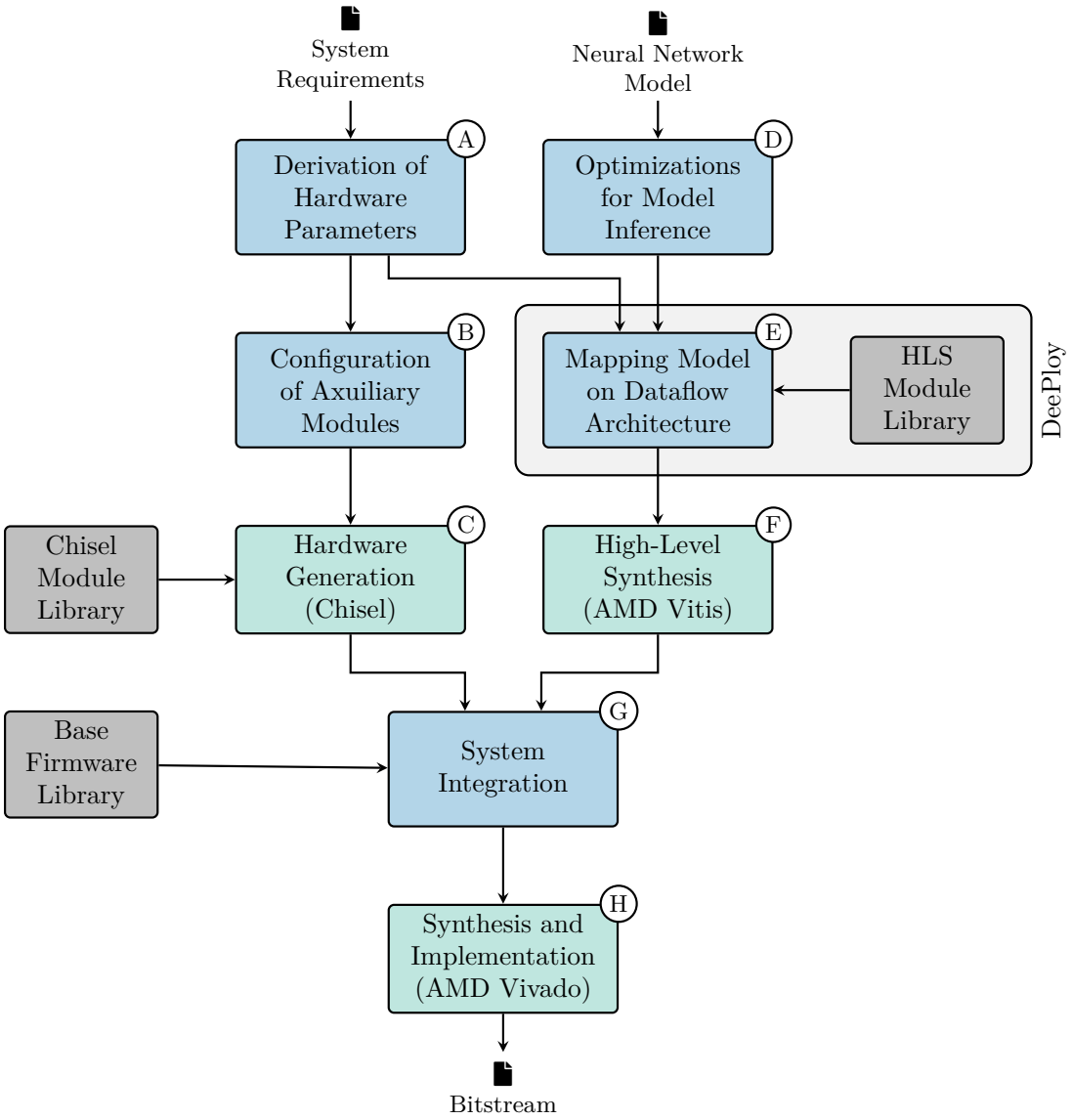


Figure 6.8: Overview of the toolflow methodology for the deployment of the CaloClusterNet onto the GNN-ETM. Design steps are labelled Ⓐ – Ⓗ and colour-coded by origin: steps shown in blue represent original contributions of this work, while steps shown in green employ existing electronic design automation tools. Supporting libraries are shown in grey. Adapted from ref .[Neu26a].

generation flow with the template-based operator mapping of DeePloy, thereby enabling consistent parameterisation across the otherwise independent hardware generation and model mapping steps.

The toolflow methodology described in this section addresses the translation of a given CaloClusterNet model into a hardware implementation on the GNN-ETM architecture. An overview is provided in figure 6.8. The methodology takes the following inputs: (i) a model description of the target GNN, containing the network topology and its pre-trained weights, and (ii) a set of system requirements as given in requirements 6.1 to 6.4.

From these system requirements, implementation-specific parameters are derived for each hardware module described in section 6.3 Ⓐ. For example, the maximum number of active TCs supported by the system is determined by the required throughput: sustaining 8 million snapshots per second with up to 32 active TCs per snapshot sets an upper bound on the parallelism of the preprocessing stage. Configuration parameters for auxiliary modules are derived in step Ⓑ: the parameters of the preprocessing stage, the postprocessing stage, and the Belle2Link Media Access Control are set such that all system requirements are satisfied, as described in detail in section 6.4.2. These configurations serve as input to both the hardware generation Ⓒ and the dataflow mapping Ⓔ, ensuring compatibility across otherwise independent design steps.

For hardware generation Ⓒ, Chisel [155] is used as the hardware construction language. Custom hardware generators were developed for key components: the preprocessing and postprocessing stages, and the Belle2Link media access control layer. These generators are configured using a YAML file produced in step Ⓑ, allowing flexible specification of design-time parameters. The resulting Verilog, produced by the Chisel toolchain, is packaged as a FuseSoC [172, 173] IP core and prepared for system-level integration Ⓖ.

Deployment of the GNN model begins with the optimisation of its inference compute graph Ⓓ, including quantisation, pruning, and algorithmic optimisations for improved hardware efficiency on the target FPGA. One such example is operator fusion, which reduces the number of intermediate buffers required between consecutive layers, directly reducing resource utilisation and latency; more details are given in section 6.4.3. The optimised inference dataflow graph is then mapped onto corresponding HLS templates Ⓔ from the custom architecture library using the DeePloy methodology (see also section 4.1). In step Ⓕ, the resulting high-level model description is converted into a synthesisable RTL description using AMD Vitis HLS 2024.2 [156].

Once RTL descriptions for all individual modules are available, the system is integrated into the base firmware Ⓖ. The complete firmware is then synthesised, placed, and routed using AMD Vivado 2024.2 [167] Ⓗ. Physical optimisations, such as manual floorplanning and the selection of synthesis parameters, are applied to the full design, as described in section 6.4.4.

6.4.2 Sparsity Compression

The GNN-ETM is provided with a snapshot of the ECL detector. Each snapshot is defined by a trigger window that contains information on all $n_{\text{TCs}} = 576$ TCs in the detector. However, due to the $\mathcal{O}(n_{\text{TCs}}^2)$ complexity of the CaloClusterNet algorithm, it is not feasible to process all TCs in each data window (requirement 6.3). A sparsity compression is therefore implemented in the preprocessing stage of the GNN-ETM to reduce the computational load of the successive GNN dataflow accelerator (see also section 6.3.1). This subsection details the dimensioning of the sparsity compression mechanism to determine the design parameter $n_{\text{TCs}}^{\text{max}}$: the maximum number of active TCs that are passed to the GNN accelerator.

Figure 6.9 shows a histogram of the input density $\frac{n_{\text{TCs}}}{576}$ measured at the GNN-ETM during the highest luminosity record at Belle II in 2024. The histogram shows that the vast majority of trigger windows have a density below 5.55 %, as indicated by the red line. Only a small fraction of trigger windows exceeds this threshold.

The sparsity compression module described in section 4.6, implemented in the GNN-ETM preprocessing stage, transforms the input matrix of n_{TCs} rows into a deterministically sized output matrix of $n_{\text{TCs}}^{\text{max}}$ rows. Based on the observation above, the design parameter $n_{\text{TCs}}^{\text{max}} = 32$ is derived, corresponding to a maximum input density of 5.55 %. This parameter directly governs the throughput of the successive GNN dataflow accelerator stage, since $n_{\text{TCs}}^{\text{max}}$ sets the number of TC features to be processed per trigger window. It thereby determines the minimum processing time per window. Substituting $n_{\text{TCs}}^{\text{max}} = 32$ into equation (6.12) yields a lower bound on the required system clock frequency f_{sys}, as shown in equation (6.13).

$$f_{\text{sys}} \overset{!}{\leq} \left\lceil \frac{n_{\text{TCs}}^{\text{max}}}{p} \right\rceil R_{\text{th}} \tag{6.13}$$

Inserting $R_{\text{th}} = 8\,\text{MHz}$ from requirement 6.3 yields feasible solutions for several values of the parallelism parameter p. For $p = 1$, one obtains $f_{\text{sys}} = 254.44\,\text{MHz}$. For $p = 2$, the

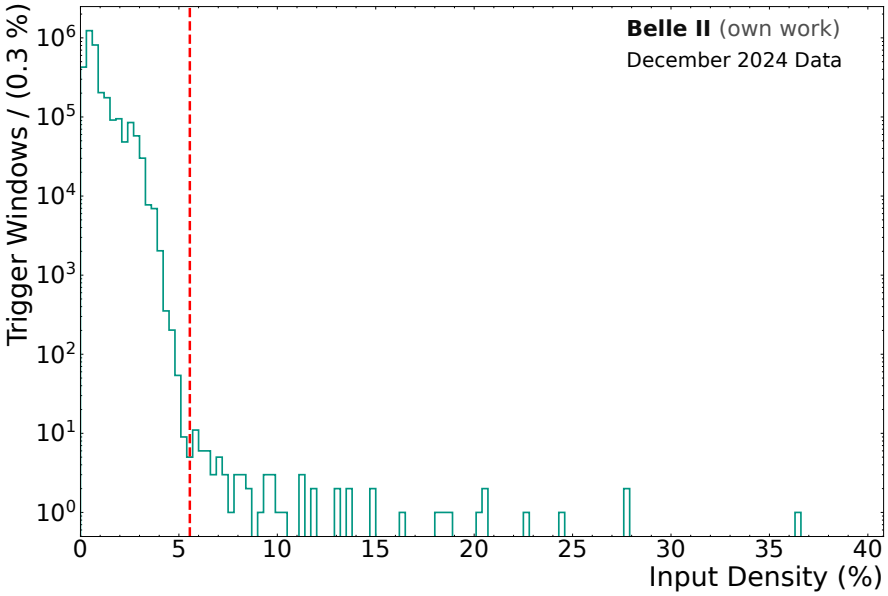


Figure 6.9: Histogram of input density for TCs received at the GNN-ETM during the highest luminosity record at Belle II in 2024. The red line marks the 5.55 % input density threshold. The y-axis depicts the logarithmic distribution of events for these runs, with equally sized bins of width 0.3 %. The x-axis depicts the input density, calculated as the number of active TCs per trigger window divided by N_{TCs}.

equation yields $f_{\text{sys}} > 127.22\,\text{MHz}$, which relaxes the frequency constraint at the cost of higher FPGA system resource utilisation. This coincides with the design frequency of the Belle II L1 Trigger system and motivates choosing $p = 2$ as the baseline design point.

6.4.3 Mapping of the Neural Network

The training of the network has been studied in refs. [1, 174]. All inference-only optimisations performed in this section are original work.

For deploying the pretrained, quantised CaloClusterNet (see section 6.1) onto an FPGA, DeePloy (see chapter 4) is used as the toolchain. The toolchain facilitates the mapping of neural network layers onto HLS architecture templates optimised for real-time inference of GNNs. Every layer is mapped onto one or more PEs. For clarity, PEs are named the same as the operators they implement. Generally, the communication between PEs is facilitated by AXI-Streams in which all features are concatenated. At parallel branches in the dataflow graph, topology PEs such as multicast and concat operators are inserted. This transformation

is required because AXI-Streams must be instantiated in parallel to enable concurrent read access to the same underlying tensor data. Reuse of the same data is not permitted. As all PEs operate independently and their control flow is fully data-driven, the realised accelerator is referred to as a dataflow accelerator. In the following, two versions of the CaloClusterNet, the Radiant Armadillo and the Sunset, are described.

The result of the mapping facilitated by DeePloy for the Radiant Armadillo CaloClusterNet is shown in figure 6.10. Radiant Armadillo constitutes the first neural network deployed on the GNN-ETM. The figure shows the dataflow graph of the accelerator after the mapping stage, consisting of 23 PEs. Green blocks represent PEs on the FPGA, whereas blue blocks indicate hierarchical containers comprising multiple PEs. This abstraction layer is introduced for better clarity. All PEs are given an identifier mapping to a layer of the original neural network. Hyperparameters for all layers are described in more detail in appendix A.1.

The following provides an overview of the resulting accelerator architecture: The input features (Timing, Energy, Position) are concatenated and transmitted as an AXI-Stream. For dense layers, matrix–vector multiplication, activation function, and rescaling are fused into a single PE. Linear layers correspond to dense layers without an activation function. Output layers differ in their activation: they either contain no activation function or a sigmoid activation function. The `mult` PE multiplies the energy factor output with the input energy, ensuring that the network returns a calibrated cluster energy in GeV. All output features are concatenated into an AXI-Stream.

A second version of the CaloClusterNet, named Sunset, is shown in figure 6.11. This neural network is a reduced version of Radiant Armadillo, which has only 16 layers in total. Most importantly, this version of the network has only one GravNet layer, which is the main driver of design complexity. In addition, the quantisation of this network has been adjusted through careful training. A maximum resolution of 8 bit is used for all layers, compared to up to 16 bit in the Radiant Armadillo model. Hyperparameters for all layers are described in more detail in appendix A.2.

In addition to the default DeePloy toolflow, a series of additional optimisations are applied. These optimisations aim to reduce the computational complexity of the CaloClusterNet to improve system performance on FPGA. At the algorithmic level, three optimisations are applied.

First, the exponential $y = e^{-|10x|}$ function used in the GravNet layer is replaced by an approximate version stored in a LUT during design time. The LUT covers the interval $x \in [0, 0.5]$,

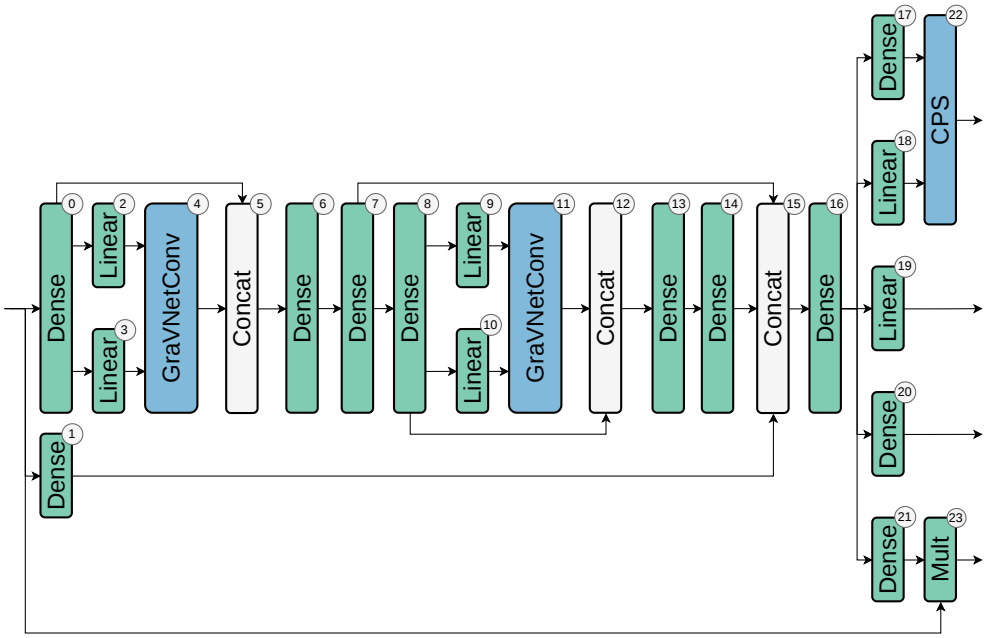


Figure 6.10: Mapping of Radiant Armadillo CaloClusterNet onto the dataflow architecture. A detailed overview of all hyperparameters is given in appendix A.1.

while values with $x > 0.5$ are mapped to zero. A LUT with 256 entries and 8 bit resolution is chosen for a similar numerical precision in comparison to other layers in the CaloClusterNet model. Figure 6.12a shows the difference between the exact and the approximated function for $x \in [0,1]$. The approximated function does not require any floating-point operation and achieves an absolute error below 0.015 for all entries. To ensure that the remaining error influences the result as little as possible, the approximated function is included in the quantisation-aware training flow [174].

Second, the sigmoid $y = (1+e^{-x})^{-1}$ function used in the output layers of the network is replaced by a hard sigmoid approximation. The same approximation as presented in ref. [130] is adopted to reduce resource usage and latency for this activation. Figure 6.12b shows the difference between the exact and the approximated function for $x \in [-6,6]$. For better comparability, the approximated version is shown with a uniform fixed-point quantisation with 8 bit precision (Q2.6).

Third, the required bitwidths of internal registers, such as accumulators, are determined to prevent both overflow and truncation of results. As an example, for a dense layer with input dimension d_i, the required accumulator width w_{acc} can be calculated as given in equation (6.14). w_{weight}, w_{bias}, w_{input} are the bitwidths for weights, biases, and inputs,

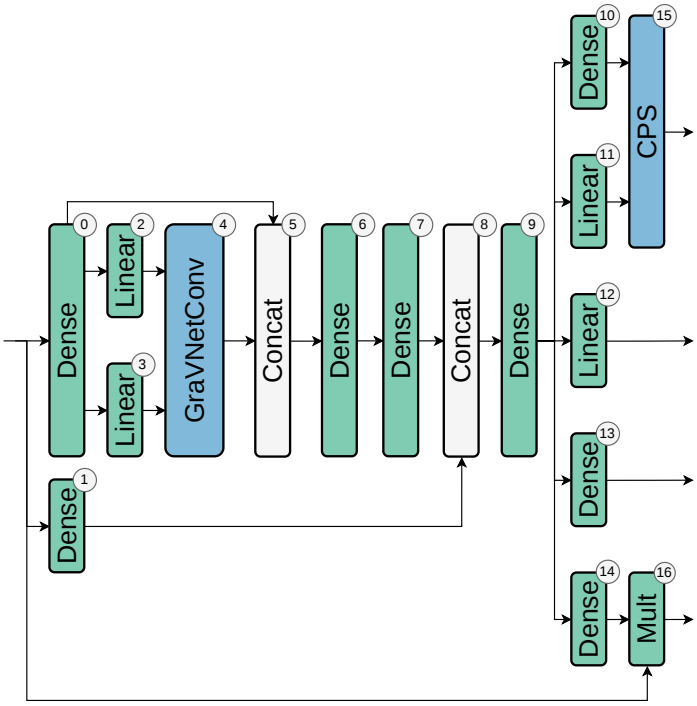
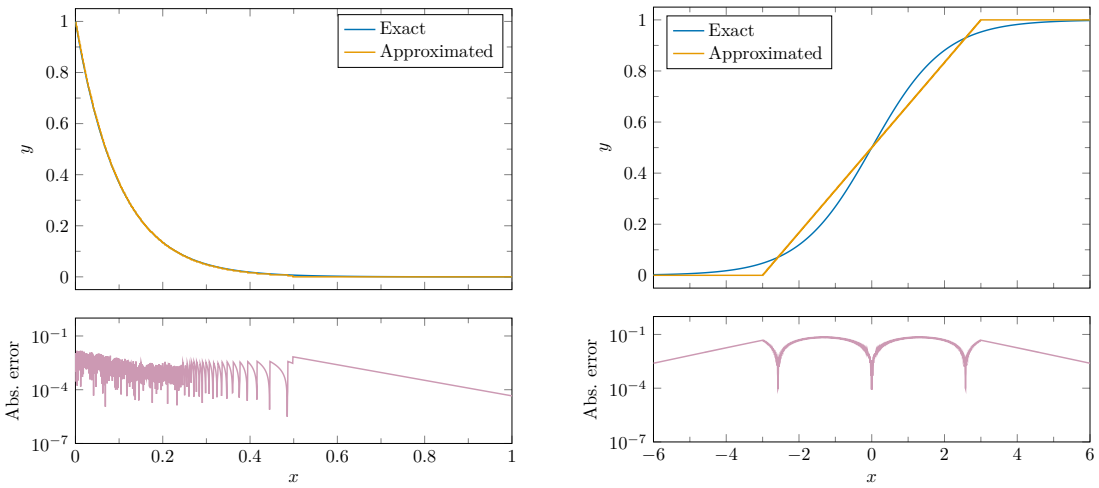


Figure 6.11: Mapping of Sunset CaloClusterNet onto the dataflow architecture. A detailed overview of all hyperparameters is given in appendix A.2.

respectively. All accumulator sizes are chosen based on such conservative, worst-case static bounds computed at design time.

$$w_{acc} = \max(\lceil \log_2(d_i) \rceil + w_{weight} + w_{input}, w_{bias}) \tag{6.14}$$



(a) Exponential function.

(b) Hard sigmoid.

Figure 6.12: Algorithmic approximations used for the exponential function and the hard sigmoid in the CaloClusterNet inference.

Beyond the algorithmic optimisations described above, a series of hardware-level optimisations is applied by configuring the HLS design tool. While AMD Vitis HLS 2024.2 is used for this implementation, the optimisation could also be applied to other HLS tools.

- The depths of AXI-Stream buffers between PEs are optimised to sustain stall-free pipelines, maximising throughput at minimal overhead in terms of LUTs and RAM. Correct sizing of these buffers is an NP-complete problem [175] and therefore requires simulation to determine the minimal sufficient depth for each stream.
- Only flushable pipeline styles are used, as the design operates in free-running mode and the additional control-flow overhead of non-flushable pipelines is unnecessary.
- The dataflow optimisation is configured to disable start propagation and return signalling, thereby eliminating residual control-flow overhead in the generated RTL.
- The number of reset register stages is increased to distribute reset fanout across additional pipeline registers, thereby aiding place-and-route. Reset signals are inserted

only for control-flow paths.

- The implementation frequency is overconstrained to 350 MHz which is significantly higher than the final target frequency. As a result, AMD Vitis HLS infers additional pipeline registers, thereby introducing greater freedom for the place-and-route algorithms in subsequent implementation steps.

6.4.4 Physical Optimizations

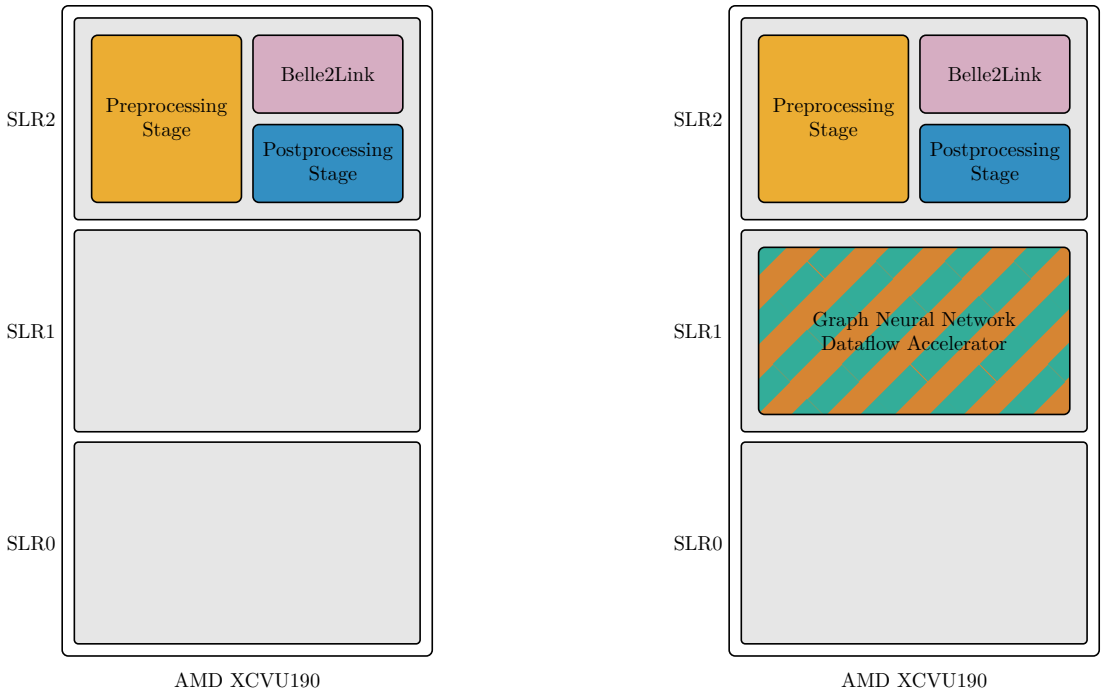
Closing timing on FPGA designs is nontrivial, particularly for large designs spanning multiple SLRs and at high system frequencies such as the 254.44 MHz clock used in the GNN-ETM. The following describes a series of optimisations applied at the RTL and physical implementation levels. Synthesis, place, and route are performed using AMD Vivado 2024.2. All optimisations therefore refer to the intrinsics of that electronic design automation toolchain.

Reset network optimisation. The reset network of the GNN-ETM is separated into two stages. A global stage distributes the reset signal to each module asynchronously to avoid routing congestion. The second stage is local to each module, where a reset synchroniser is implemented with a fixed number of pipeline stages. This ensures that the reset for large modules, such as the GNN dataflow accelerator, is properly distributed across multiple SLRs without influencing timing closure. By deasserting the reset signal synchronously, race conditions are avoided. This optimisation is applied to both designs.

Tool directives. AMD Vivado provides numerous synthesis and implementation directives. For all designs, `Flow_PerfOptimized_High` is selected for synthesis, and global retiming is enabled to exploit additional register stages. For implementation, all strategies in AMD Vivado 2024.2 are evaluated to identify the best set of optimisation options. For the design with the Radiant Armadillo CaloClusterNet, `Congestion_SpreadLogic_high` ↪ yields the best timing-closure results. For the design with the Sunset CaloClusterNet, `Performance_Auto_3` yields the best timing-closure results.

Manual floorplanning. Automated placement and routing for multi-SLR designs presents a difficult challenge [176]. This is especially true when strict external constraints are enforced. In the case of the GNN-ETM, all gigabit transceiver ports are grouped into SLR 2, and input and output modules are therefore naturally placed there. Figure 6.13 depicts the floorplan concepts for the two design instances under evaluation. For the Radiant Armadillo design, no constraints are imposed on the GNN dataflow accelerator because its size exceeds two

SLRs. For the Sunset design, the GNN dataflow accelerator is placed on SLR 1 for two reasons: First, one SLR provides sufficient resources for the complete dataflow accelerator. Second, placing the accelerator on a separate SLR from the remaining modules eases timing closure, as utilisation on SLR 2 is reduced. This optimisation was not automatically detected by AMD Vivado and required manual insertion.



(a) GNN-ETM with the Radiant Armadillo CaloClusterNet model.

(b) GNN-ETM with the Sunset CaloClusterNet model.

Figure 6.13: Floorplan constraints of the GNN-ETM for implementation with AMD Vivado 2024.2. Hierarchical modules are the same as in figure 6.3. Hierarchical modules that do not appear in the floor plan are not subject to any location constraints. Adapted from ref. [Neu26d].

Disable DSPs. Disabling DSPs has proven effective for mitigating routing congestion and placement errors. By setting the option `-max_dsp=0`, Vivado is prevented from mapping any multiplication to dedicated processing units. While this generally increases resource utilisation on LUTs, it aids placement as DSPs are sparsely scattered across the fabric. This is especially true in designs that lack sufficient registers to span the wide distances between DSPs on chip. When the fixed-point precision is sufficiently low, a positive effect on timing closure has been observed. In the presented design study, precisions of 8 bit or less have proven effective. This optimisation is applied only to the GNN-ETM with the Sunset CaloClusterNet.

The combined effect of these optimisations on timing closure, resource utilisation, and system latency is evaluated in section 6.7.

6.5 Software

This section describes the software components implemented to interface with the GNN-ETM. Section 6.5.1 describes the GNN-ETM Hardware Abstraction Layer (HAL) and its available functionalities. Section 6.5.2 describes the integration of the GNN-ETM into the global Belle II slow control via NSM 2. Section 6.5.3 briefly describes the unpacker for Belle2Link packets generated by the GNN-ETM.

6.5.1 Hardware Abstraction Layer

Low-level details of the GNN-ETM software interface, such as register addresses, are abstracted in a HAL developed for this purpose. This HAL encapsulates all register-level access in a C++ class. The register map is defined in a SystemRDL description and compiled to a packed C struct hierarchy using PeakRDL [177]. The generated layout is verified at compile time using a static assertion against the firmware address map. An overview of all registers is given in appendix C. Internally, the `GNN` class uses `volatile` pointer on the address range of the device exposed in the Linux driver for the VME bus. The most important public methods are summarised in table 6.5.

All configurable parameters are consolidated into `gnn_config_t`:

```
typedef struct {
    uint32_t data_delay; // Belle2Link data delay
    uint32_t trg_delay; // Belle2Link trigger delay
    gnn_trigger_line_t trigger_line; // Trigger bit via LEMO
    uint32_t ccn_beta; // CaloClusterNet beta threshold
    uint32_t ccn_signal; // CaloClusterNet signal threshold
    uint32_t ccn_isolation; // CaloClusterNet isolation threshold
} gnn_config_t;
```

Listing 6.1: Configuration struct for the GNN-ETM.

Configuration registers should be written only through the public `WriteConfig()` method, never directly to the registers. For configuration settings to be applied successfully, the GNN-ETM must first be brought into an idle state. `WriteConfig()` also performs a readback of

Table 6.5: Public interface of the GNN HAL class.

Category	Method	Description
Identification	`ReadFirmwareIdentifier()`	Reads the firmware identifier string.
	`ReadFirmwareGitHash()`	Reads the firmware git hash.
Control	`Reset()`	Hard reset of the full system.
	`Clear()`	Clears all monitoring registers.
	`Reboot()`	Power reboot of system via auxilary FPGA onUT 4
Status	`ReadICNRxLinkStatus()`	Returns transceiver link status on the incoming interface from ICN-ETM.
	`ReadGDLTxLinkStatus()`	Returns transceiver link status on the outgoing interface to GDL.
	`ReadB2LStatus()`	Returns transceiver link status on the outgoing interface to Belle2Link.
	`ReadClockCounters()`	Reads system, data, and gigabit transceiver clock counters.
	`CalculateClockRates()`	Derives clock frequencies from two clock counter snapshots.
Monitor	`ReadMonitor()`	Reads trigger bit monitoring counters.
	`CalculateMonitorRates()`	Derives trigger rates from two counter snapshots.
Configuration	`ReadConfig()`	Reads current configuration into `gnn_config_t`.
	`WriteConfig()`	Writes `gnn_config_t`.

all configuration registers to verify that each value has been applied correctly. If a mismatch is detected, an assertion error is thrown.

6.5.2 Slow Control Integration

Figure 6.14 shows an overview of the slow control architecture at Belle II for the ECL L1 Trigger at Belle II. ICN-ETM and GNN-ETM are both connected via VME bus to the VME CPU server. This server is directly connected to the UT 4 trigger boards, thus enabling direct hardware access. The GNN-ETM is mapped to the base address `0x71000000`, whereas ICN-ETM is mapped to the base address `0x51000000`.

To enable communication with the global slow control system at Belle II, the VME CPU hosts an NSM 2 endpoint [44]. From here on, communication is facilitated through the NSM 2

network. The trigger control server `btrgctrl0` interfaces with this endpoint and forwards it to the remaining trigger slow control network.

As a final step, the endpoint is made available to the global run control system, which orchestrates the operation of the complete Belle II experiment. Process variables are additionally exposed via the NSM 2 node and, after registration in the database, continuously logged by the EPICS archiver [178]. Polling-mode logging is used for GNN-ETM with a rate of 1 Hz for all process variables.

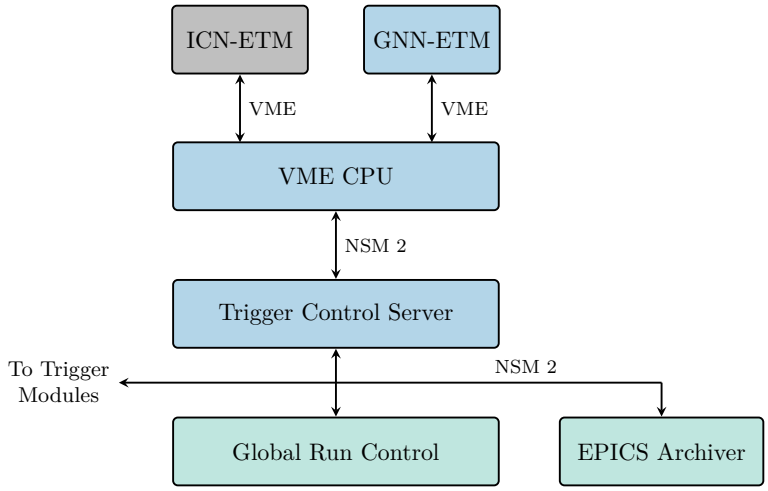


Figure 6.14: Overview of the GNN-ETM slow control at Belle II, extended from ref. [44]. Systems which were modified to support GNN-ETM are shown in blue. Global slow control modules are green. ICN-ETM is included as a reference in grey.

The GNN-ETM slow control node is implemented as an NSM 2 callback node, integrating the HAL with the Belle II run control and EPICS archiver infrastructure. Hardware status, configuration parameters, and trigger bit monitoring counters and rates are published as NSM 2 variables, which are mapped to EPICS process variables via a `.db` record file loaded at startup. Each process variable is bound to an NSM 2 variable handler via a dedicated device type, as shown for the `ready_signal` status variable in listing 6.2.

```
record(stringout, "B2_nsm:set:ECLTRG_GNN:ready_signal")
{
  field(SCAN, "Passive")
  field(DTYP, "nsm2_vset_stringout")
}
```

Listing 6.2: Example EPICS record definition for the GNN-ETM ready signal.

The Finite State Machine (FSM) of the GNN-ETM slow control node is shown in figure 6.15. After boot, the system initialises into the `NOTREADY` state. When an experiment run is

initialised, the LOAD transition is executed on GNN-ETM, selecting the parameters from the configuration file based on the current mode (either `[local_run]` or `[global_run]` ↪) before writing all configuration registers. LOAD writes all configuration registers from the configuration database into the respective registers, as well as resetting all monitoring registers. An example configuration database file is shown in listing 6.3. When a LOAD command is received from the NSM 2 node, a binary flag indicates the current run mode of the experiment. As soon as this transition is complete, the state is set to READY. The GNN-ETM is now armed and waits for input signals. When all subtrigger modules indicate their READY status to the experiment operator, the START transition is executed by the experiment operator. At any point, the operator may terminate the run via STOP and return to NOTREADY via ABORT, which resets the trigger pipeline and clears all monitoring registers. In case of a detected fault, the FSM automatically transitions into the ERROR state, from which recovery requires manual intervention via ABORT.

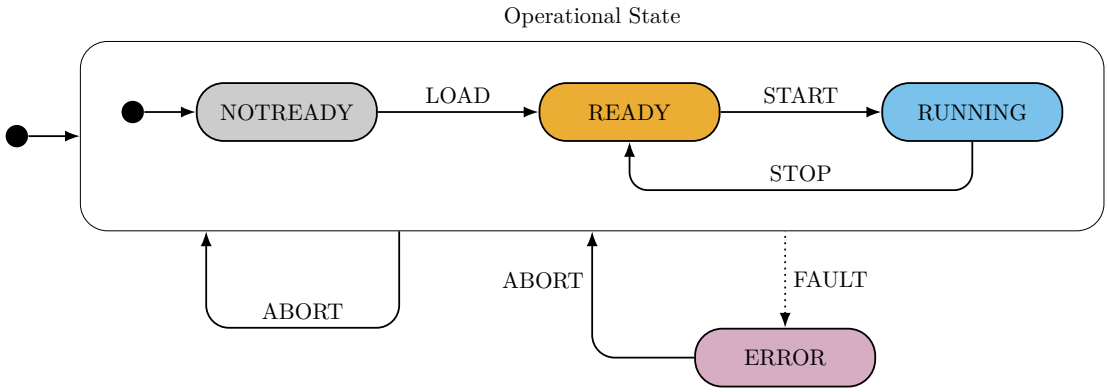


Figure 6.15: Unified Modeling Language FSM of the GNN-ETM slow control node. State and transition definitions are based on the global state machine from ref. [44]. Colours are chosen to match the Belle II run control interface for the experiment operator. Dotted lines indicate automatic transitions.

To simplify the reset procedure for L1 Trigger expert shifters during operation, specific command chains are provided: `recover`, `reboot`, and `freboot`. Because the GNN-ETM slow control node is integrated with the ICN-ETM slow control node, both systems can be reset only together. When a `recover` command is issued to the GNN-ETM slow control node, first ABORT and then LOAD transitions are executed. Effectively, this command reloads all configuration parameters and clears monitoring registers. This is the fastest recovery method and requires only a minimal set of register writes. The `reboot` command issues a full reset of the GNN-ETM core logic. Afterwards, all configuration registers must be rewritten during the LOAD transition. This complete reset takes less than 1 s. The `freboot` command forces

reprogramming of the GNN-ETM firmware on the UT 4 board. This last-resort command triggers a complete firmware reload and takes approximately 30 s.

```
[local_run]
data_delay    = 0
trg_delay     = 33
trigger_line  = GNN_CPT_C2_100MEV
ccn_beta      = 0xEE66
ccn_signal    = 0xE66
ccn_isolation = 0x100

[global_run]
data_delay    = 8
trg_delay     = 17
trigger_line  = GNN_CPT_C2_100MEV
ccn_beta      = 0xEE66
ccn_signal    = 0xE66
ccn_isolation = 0x100
```

Listing 6.3: Example configuration file for the GNN-ETM, showing parameters for local and global run modes.

6.5.3 Belle2Link Unpacker

The integration of the Belle2Link Unpacker into the Belle II Analysis Framework is described in ref. [1].

A software unpacker decodes GNN-ETM Belle2Link packets and makes the trigger information they contain available for verification and offline analysis. For validation in RTL simulation, a standalone version, independent of the Belle II Analysis Framework in C, is developed. Due to the variable packet sizes, decoding the information is nontrivial and must be handled carefully. Its signature is shown in listing 6.4.

```
int unpack_gnnetm_event(uint8_t *event_ptr, size_t length, bool
   ↪ verbose, int16_t *cells, int16_t *clusters, int8_t  *
   ↪ triggerlines, int *ids, int *num);
```

Listing 6.4: GNN-ETM event unpacker.

The unpacker consumes a raw byte buffer of the declared length in bytes and writes the decoded trigger cells, clusters, GDL trigger bits, and cell identifiers into caller-supplied output arrays. In addition, the method returns the number of integrity violations encountered during decoding, enabling detection of malformed events in the simulation or when unpacking real event data.

Decoding follows the layout of the output packet. The header is parsed first to obtain the firmware version and revolution clock, followed by the per-window headers and the bitmask generated by the condensation point selection algorithm. Next, the GDL frame is processed, yielding event time, cluster count, and all trigger bits per trigger window. Finally, the cell and cluster payloads are extracted. The hardware-level packing order is inverted during extraction so that downstream software observes natural index ordering.

Several cross-checks are applied during decoding. First, the monotonic progression of the FAM clock counters across windows is validated. Second, TC identifiers are checked for their legal range. Third, the alignment of the bitmask by the condensation point selection algorithm is checked. Fourth, consistency between the calculated packet length based on packet header information and the actual packet length is checked. The method returns the total number of structural violations encountered during packet decoding.

The Belle2Link unpacker is used in section 6.6 to validate the GNN-ETM in simulation. A complete overview of the packet structure is given in appendix B.

6.6 Validation

The correct functionality of the GNN-ETM is verified through comprehensive simulations across multiple abstraction levels. An overview of the simulation framework is provided in figure 6.16. Based on either technical input samples, simulated data or previously recorded events, simulations are conducted on three levels of abstraction:

Quantised model simulation. Reference results are computed using the quantised model implemented in QKeras. This simulation is also used during quantisation-aware training.

C-level transaction simulation. This simulation, implemented in AMD Vitis 2024.2 [156], evaluates neural network inference and the condensation point selection algorithm. It only covers the GNN dataflow accelerator mapping the CaloClusterNet model, since this is the only module designed using HLS.

Cycle-accurate RTL simulation. This simulation is conducted for the GNN-ETM core logic, excluding only hard IP cores such as gigabit transceivers and clocking resources. Input events

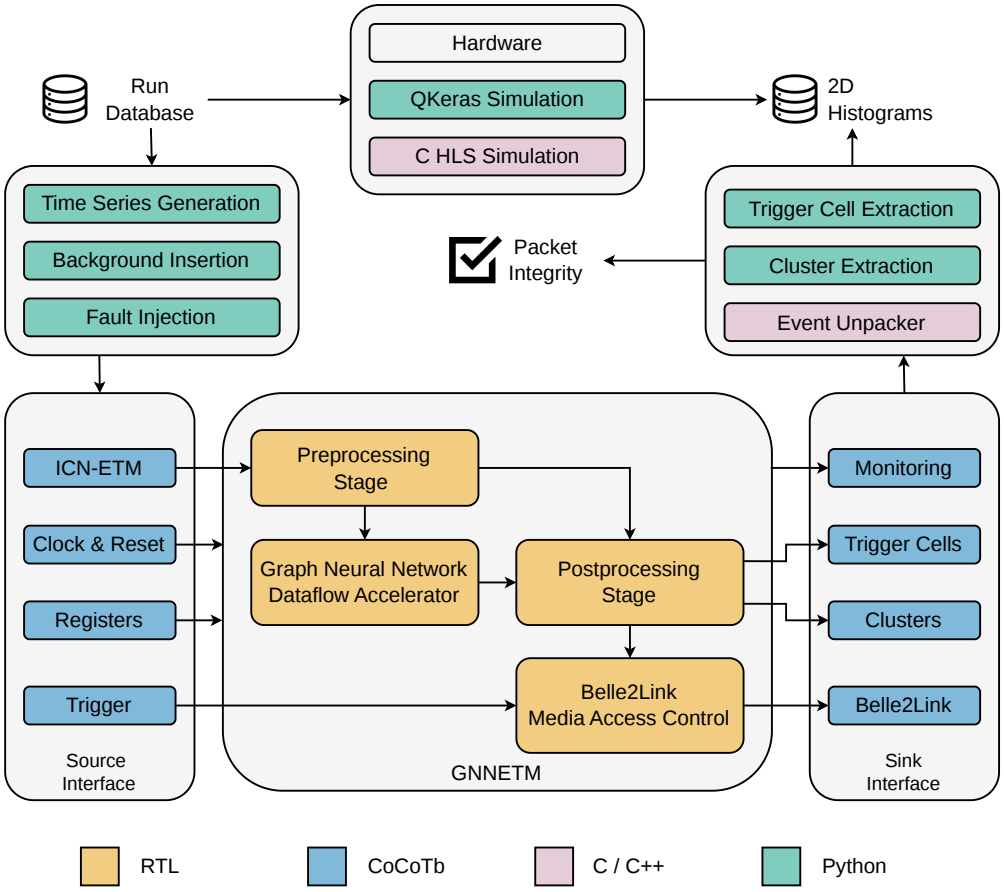


Figure 6.16: Full simulation framework for the GNN-ETM validation. The workflow for comparing the simulations across all three levels of abstraction and the hardware output itself is shown, along with the respective software used for implementation. Adapted from ref. [Neu26a].

from the run database are converted into time-series data streams compatible with the AXI-Stream interface. The simulation is executed in ModelSim 2023.4 [169] with CoCoTb [179] and the CoCoTb-AXI [180] extension. The Belle2Link subsystem driver replicates the behaviour of the Belle2Link protocol. Packets received by the driver are unpacked with a C implementation of the experiment's software unpacker, enabling complete end-to-end validation of the processing chain.

In the following, two categories of validation are described: Section 6.6.1 covers system-level validation, ensuring that the firmware behaves as intended, while section 6.6.2 covers data-level validation, ensuring that the algorithmic results computed by the firmware are bit-identical to the offline algorithmic model of the system.

6.6.1 System-Level Validation

To validate the GNN-ETM at the system level, a set of testbenches with test cases is defined and described in the following. A central challenge of RTL validation is simulation duration: simulating 90 recorded events requires, on average 1 h of wall-clock time, while covering only 3 ms of experiment time at a trigger rate of 30 kHz. Rarely occurring faults are therefore difficult to detect and reproduce at this level.

The following five test cases are implemented:

Control Test. This test applies stimuli to the control registers, including the trigger and data delay, of the Belle2Link Media Access Control module (see also section 6.3.4). This test case verifies that the most important control registers on the firmware are wired correctly and that the reset functionality operates as intended, ensuring reliable configurability of the system.

Latency Test. This test uses random TCs as stimuli. It measures the required latency for all time-critical stages in GNN-ETM: the preprocessing stage, the dataflow accelerator, and the postprocessing stage. This test case verifies that each stage meets its latency budget and provides the measurements used for the performance analysis in section 6.7.

End-to-End Test. This test applies random TCs as stimuli and asserts the trigger input at random timestamps during the simulation. A predetermined set of internal signals between the major modules is monitored during the simulation. Belle2Link packets generated by the firmware are stored as hexadecimal data in a text file, and their consistency is validated at the end of the simulation. This test case verifies that no deadlock occurs during regular, non-fault

operation for randomly distributed input data. Additionally, it provides the reference outputs used for the data-level validation in section 6.6.2.

ICN-ETM Fault Test. This test runs the same stimuli as the end-to-end test. However, after a random short time interval, faulty data is sent via the ICN-ETM to GNN-ETM transceiver interface. This test case verifies that the fault is detected as such and the GNN-ETM enters a safe error state without interfering with data taking during the experiment runs, which is particularly important as the connection between FAM and ICN-ETM has proven to exhibit faults in operation occasionally.

Beam Injection Test. Beam background injection at Belle II leads to a large number of TCs being registered in the ECL within a short time window. This scenario is reproduced by generating data windows with a large number of TCs as stimuli for a short duration. This test case verifies that the system behaves correctly under realistic beam conditions, which is particularly important as such conditions are not present during cosmic runs.

6.6.2 Data-Level Validation

To validate GNN-ETM at the data level, the C-based HLS simulation is compared with recorded data under two operating conditions: cosmic-ray events and beam-collision events. In general, validation studies are repeated after every modification to the GNN-ETM firmware to ensure correct functionality for the trigger. The network's large depth makes it sensitive to small perturbations, and even a single-bit change in 16 bit fixed-point inputs can substantially alter the final predictions. This is particularly relevant for the regression targets, where small deviations may alter the selection of the condensation point or degrade the quality of the energy and position estimates. Bitwise agreement between implementations is, therefore, a meaningful validation criterion.

In the following, the validation procedure is described for two representative implementations. The first implementation concerns GNN-ETM with the Radiant Armadillo CaloClusterNet (for details on the model see appendix A.1) and is based on 10 000 cosmic-ray events recorded on the hardware. The second implementation concerns GNN-ETM with the Sunset CaloClusterNet (for details on the model, see appendix A.2) and is based on two beam-collision datasets of 1000 events each, recorded before and after the fix described below.

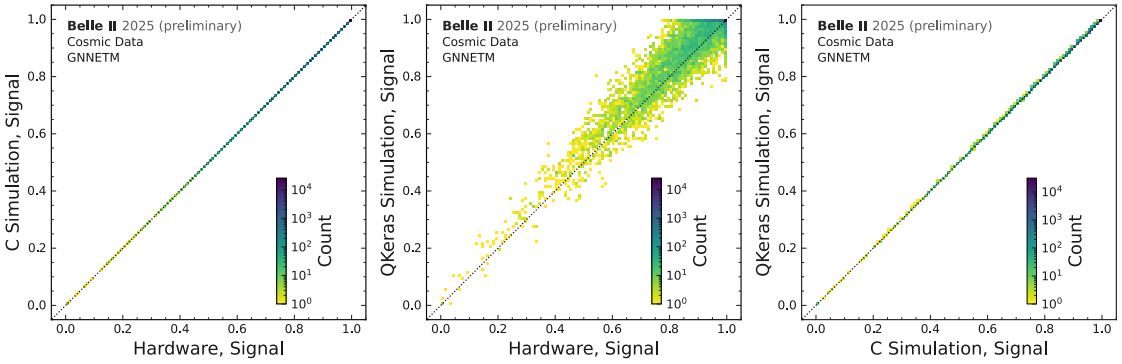
First implementation: Radiant Armadillo CaloClusterNet. The first validation is based on 18 325 trigger windows in total, since multiple trigger windows are available per event and only those containing at least one TC are retained. The comparison, shown in figure 6.17a, demonstrates bitwise agreement between the C-simulation and the hardware on the given cosmic dataset. It is therefore concluded that the C-simulation model provides a reliable basis for algorithmic performance analysis.

With the C-simulation established as a bit-accurate reference for the hardware on the given cosmic dataset, it is then used to validate the QKeras simulation. QKeras is essential for fast iteration, as it enables rapid testing of design changes and retraining of models without repeating the time-consuming hardware implementation. Because data types, rounding, and quantisation must be configured manually, the initial agreement with the hardware was poor, as shown in figure 6.17b, limiting the reliability of performance studies.

To improve this agreement, both the hardware implementation and the QKeras model were adapted. Input feature scaling was originally fused into the first dense layer, which caused a mismatch. Moving the scaling into the preprocessing stage (section 6.3.1) removed one quantisation step and resolved this issue. Truncation of intermediate results in the processing elements was also corrected by adjusting bit widths. On the QKeras side, floor rounding in the inference step is used to match hardware behaviour, and a dedicated function was implemented to model the LUT-based exponential function (section 6.4.3). The improved agreement between QKeras and the C-simulation is shown in figure 6.17c. Since the C-simulation and the hardware are in bitwise agreement on this dataset, the improved agreement also propagates to the comparison between QKeras and the hardware.

The remaining differences stem from the unstable sorting implementation. Although the same input order yields deterministic results, that ordering cannot be consistently propagated between QKeras and the other two implementations. Employing a stable sorting algorithm would enforce exact agreement across all implementations. Still, the current approach is retained as the observed differences have a negligible impact on the physics performance results.

Second implementation: Sunset CaloClusterNet. Each of the two datasets yields 990 trigger windows after retaining only those containing at least one TC. During the deployment of the improved Sunset CaloClusterNet model, a large discrepancy was observed in validation, as shown in figure 6.18a. In addition, the same discrepancy was observed when reproducing the error in RTL. The root cause was identified as a software bug in the HLS toolchain (AMD Vitis HLS 2024.2), which is only observable after post-synthesis RTL-simulation, as shown in



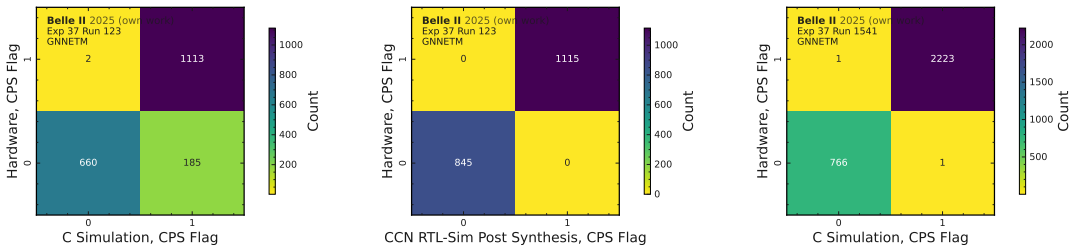
(a) C-Simulation vs. Hardware (b) QKeras vs Hardware, before (c) QKeras vs Hardware, after

Figure 6.17: 2D histograms of the predicted signal classifier values from C-simulation vs hardware (figure 6.17a), QKeras vs hardware with the original agreement (figure 6.17b) and QKeras vs C-simulation after improvements to both C-simulation and QKeras (figure 6.17c). The agreement between the C-simulation and the hardware is bitwise correct for the given cosmic dataset.

figure 6.18b [181]. This finding highlights a fundamental limitation of HLS-based design flows: C-level simulation alone is insufficient to guarantee bit-accurate correspondence with the synthesised hardware, and post-synthesis RTL-simulation is required as an additional validation step. It is noted that this bug is not present across all network configurations and depends on the specific weights. Retraining the network may resolve the issue. Comparing RTL code with and without DSP primitives enabled revealed that DSP inference altered the algorithmic output, isolating the bug to the HLS tool's DSP mapping. As a workaround, all DSP inference was disabled in the firmware, which resolved the discrepancy, as shown in figure 6.18c. Starting from experiment 37, run 1541, the GNN-ETM is validated with collision data in the Belle II detector. Bitwise agreement between C-simulation and hardware is recovered on the after-fix dataset, matching the result established in the first validation implementation. The only observed difference between the C-simulation and the hardware in experiment 37, run 1541, is due to unstable sorting. More details on the operation of the system are given in section 6.8.

6.7 Performance Analysis

To determine the system performance for latency (requirement 6.2) and throughput (requirement 6.3), an RTL simulation is performed. For each design iteration, the system frequency is chosen according to the lower bound derived in equation (6.13), such that requirement 6.3 is met by construction; throughput is additionally validated for all design iterations during



(a) C-Simulation vs. Hardware (b) Post-Synth. RTL-Sim. vs. Hardware (c) C-Simulation vs. Hardware

Figure 6.18: Confusion plot of the predicted condensation point selection flag from C-simulation vs hardware (figure 6.18a), post-synthesis RTL-simulation vs hardware with the original agreement (figure 6.18b) and C-simulation vs. hardware after improvements to the hardware (figure 6.18c).

simulation. The remainder of this section, therefore, focuses on latency. End-to-end latency of the GNN-ETM architecture is reported in figure 6.19, measured from the AXI-Stream input received from ICN-ETM to the AXI-Stream output of the postprocessing stage. This path, highlighted in figure 6.3, represents the critical real-time segment within the L1 Trigger system. Obtaining latency values for each stage is a two-step process. First, the full GNN-ETM is implemented using AMD Vivado 2024.2 on the AMD Ultrascale XCVU190, with the tool directives described in section 6.4.4 to ensure that timing is met for the given design system frequency. Second, cycle-accurate RTL simulations of the GNN-ETM architecture are performed with ModelSim 2023.4 [169] with the previously determined system frequency. In total, four design iterations (①—④) are evaluated for latency:

① is the original version with the Radiant Armadillo CaloClusterNet model first presented in ref. [Neu26a]. The total end-to-end inference latency is 3168 ns, dominated by the GNN dataflow accelerator with 2052 ns, followed by the preprocessing stage with 621 ns. The condensation point selection stage has a latency of 495 ns. This latency exceeds the budget for an active L1 Trigger decision by approximately a factor of three.

To improve the overall latency, several algorithmic optimisations are applied to the design in ②. First, the dataflow of the preprocessing stage is optimised by reducing the number of pipeline stages: the original design supported up to 64 active trigger cells; this has been reduced to 32, thereby decreasing the number of preprocessing pipeline stages. Second, the Radiant Armadillo CaloClusterNet is replaced with the compressed Sunset CaloClusterNet model: the fixed-point precision is reduced to a maximum of 8 bit, and one GraVNet layer is

removed from the model. Third, the calculation of trigger bits is added to the postprocessing stage, slightly increasing overall latency. The total end-to-end inference latency is 1886 ns, dominated by the GNN dataflow accelerator with 1220 ns, followed by the preprocessing stage with 385 ns. The condensation point selection stage has a latency of 250 ns and the postprocessing stage has a latency of 31 ns.

③ doubles the system frequency f_{sys} of the GNN dataflow accelerator from 127.22 MHz to 254.44 MHz. This frequency increase was not feasible in ②, as the corresponding designs did not reach timing closure in implementation. To achieve the higher frequency, manual floorplanning (see also section 6.4.4) is required. The total end-to-end inference latency is 1151 ns, dominated by the GNN dataflow accelerator with 610 ns, followed by the preprocessing stage with 385 ns. The condensation point selection stage has a latency of 125 ns and the postprocessing stage has a latency of 31 ns.

The final design iteration is shown as ④. The only difference to the previous design is the disabling of DSPs in the HLS-based GNN dataflow accelerator module. As a result, the number of pipeline stages is reduced, significantly reducing overall latency. This is because DSP blocks require mandatory pipeline registers to meet their internal timing, whereas LUT-based multipliers at the reduced fixed-point precision can be realised with a shallower pipeline. The total end-to-end inference latency is 1053 ns, dominated by the GNN dataflow accelerator with 507 ns, followed by the preprocessing stage with 385 ns. Due to the windowing function, the theoretical minimum time for this stage is 250 ns. The condensation point selection stage has a latency of 130 ns and the postprocessing stage has a latency of 31 ns. The slight increase in the condensation point selection stage amounts to one clock cycle and is caused by the disabling of DSPs. With this, Requirement 6.2 is satisfied by a margin of 14 ns.

GNN-ETM designs with the Radiant Armadillo CaloClusterNet ① and Sunset CaloClusterNet ④, the overall system resource utilisation on the UT4 platform is analyzed. The utilisation of FFs, LUTs, DSPs, and BRAMs is analysed for the entire design and per module. To quantify the resource impact of the algorithmic optimisations, utilisation is compared between the baseline ① and the final design ④.

For ①, the overall system resource utilisation is presented in figure 6.20. The design uses approximately 51 % of the available LUTs and utilises all DSP resources. BRAM usage remains below 9 %, primarily due to the Belle2Link Media Access Control. Most DSPs are used by the trainable dense layers of the neural network, particularly those computed at 16 bit precision, which benefit most from a mapping on DSPs. In contrast, 8 bit dense layers are largely mapped to distributed logic. The GraVNetConv operator accounts for a substantial share of

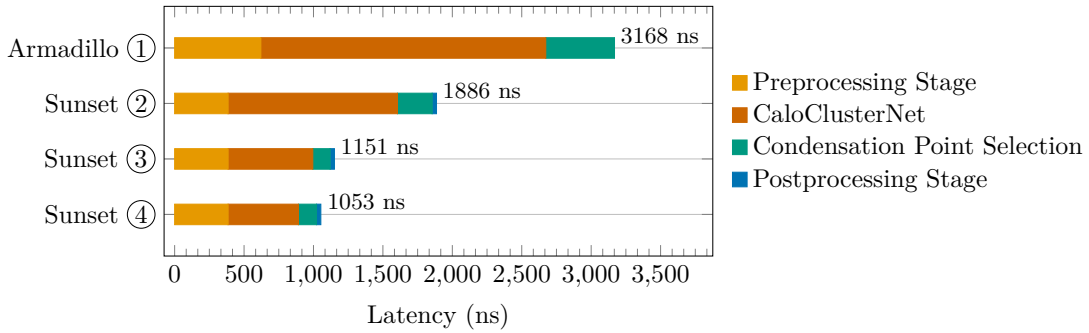


Figure 6.19: End-to-end latency for the complete inference chain on the UT4 with an AMD Ultrascale XCVU190 FPGA. Four design iterations (①—④) are presented. Taken from ref. [Neu26d].

FF and LUT usage due to its $\mathcal{O}(n^2)$ complexity in computing pairwise distances. By comparison, the Condensation Point Selection stage requires relatively few resources. Although it also computes pairwise distances, it operates in a lower-dimensional space (3 vs 6 dimensions) compared to the GraVNetConv layer. Overall, the design occupies 82.34 % of Configurable Logic Blocks (CLBs), the architecture-specific logic clusters in AMD Ultrascale FPGAs. The disproportionately high CLB usage relative to LUTs and FFs indicates routing congestion, which motivates the manual floorplanning effort applied in ③.

For ④, the overall system resource utilisation is presented in figure 6.21. The design uses approximately 23 % of available LUTs and no DSP resources. Most LUTs and FFs are for the GNN dataflow accelerator. The utilisation of LUTs is reduced from 17.59 % to 4.20 % for the GraVNet layers. This reduction is due to the lower fixed-point precision and the removal of one GraVNet layer. The utilisation of LUTs is reduced from 19.71 % to 9.36 % for the Dense layers. The added calculation of trigger bits in the postprocessing stage incurs a small utilisation penalty of 2.08 % on the LUTs. All other modules remain similar in resource utilisation.

6.8 Operation

This section has been first presented in ref. [Neu26d] and is reproduced here with minor modifications.

For operation of the GNN-ETM in the full ECL L1 Trigger chain, the system is integrated as described previously in figure 6.2. The first version of the GNN-ETM with the Radiant Armadillo CaloClusterNet has been included as a parasitic module to read out debug data

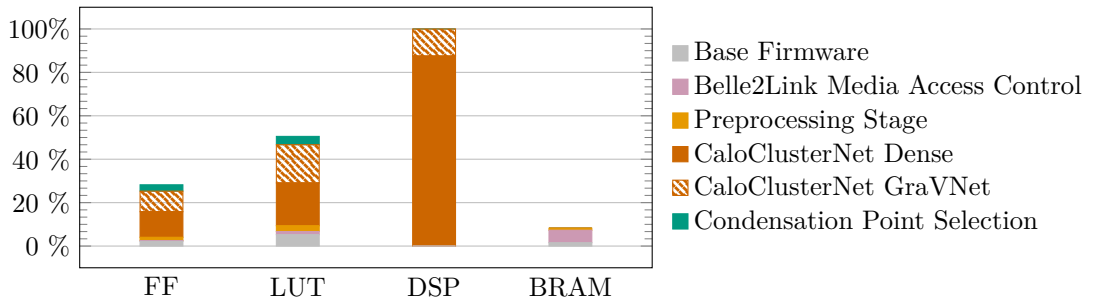


Figure 6.20: Utilisation of system resources on the UT4 with an AMD Ultrascale XCVU190 FPGA for the GNN-ETM architecture with Radiant Armadillo CaloClusterNet ①.

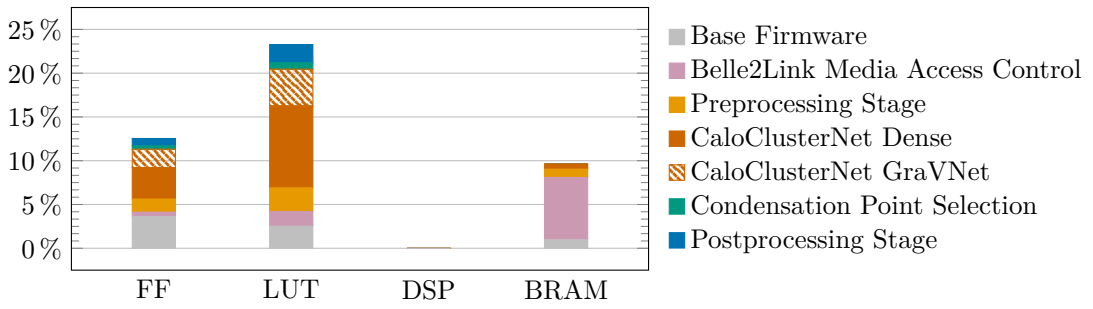


Figure 6.21: Utilisation of system resources on the UT4 with an AMD Ultrascale XCVU190 FPGA for the GNN-ETM architecture with Sunset CaloClusterNet ④.

via the Belle2Link interface [Neu26a], since the latency requirement was not yet met. With the reduction of overall latency (see also section 6.7), commissioning the GNN-ETM into the L1 Trigger system is feasible.

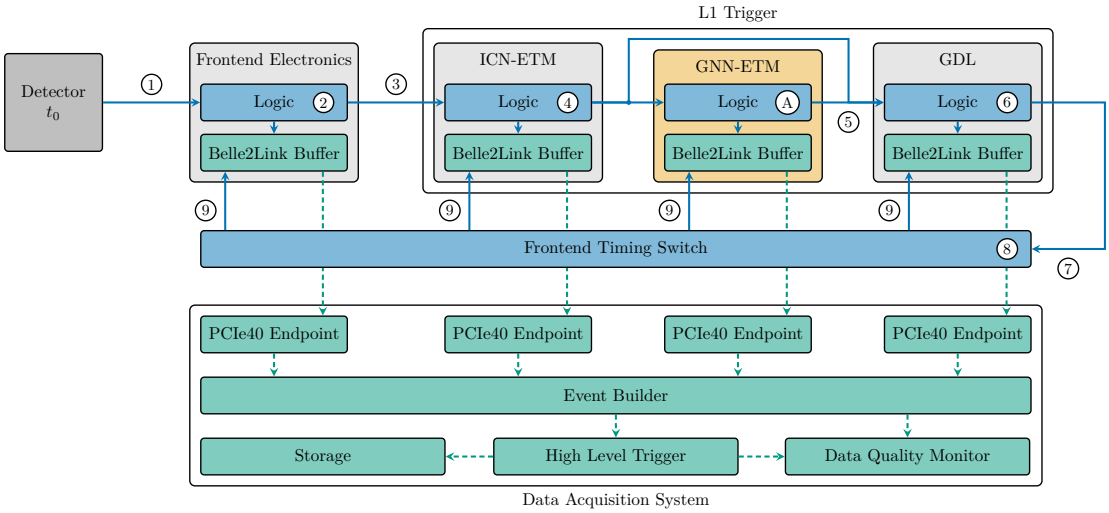


Figure 6.22: Simplified schematic overview of the data acquisition system and the L1 Trigger system at Belle II. The GNN-ETM developed in this work is highlighted in orange. Numbered circles label selected components and connections referenced in the text. t_0 denotes the time of a detected bunch crossing. Taken from ref. [Neu26d].

Figure 6.22 depicts a simplified schematic of the Belle II ECL L1 Trigger system. For clarity, the connection to the GRL, where a matching between the ECL trigger and the trigger of other subdetectors is performed, is omitted. The critical-path latency ① → ⑨ must not exceed $R_L = 5.0\,\mu s$, a constraint imposed by the finite depth of the data buffers on the frontend electronics modules. In the configuration depicted in the figure, the latency of the GNN-ETM Ⓐ must not exceed 1.067 µs (see also requirement 6.2). Switching the placement of ICN-ETM and GNN-ETM would yield an additional latency margin of 154 ns based on experimental measurements, as one GT connection is removed from the critical path.

6.8.1 Commissioning

Figure 6.23 depicts the interfaces of ICN-ETM, GNN-ETM, and GDL in the Belle II L1 Trigger system. Both ICN-ETM and GNN-ETM are connected via gigabit transceivers (GTs) to the GDL. This connection via optical fibres enables the transmission of large packets of up to 768 bit per 127.216 MHz system clock cycle, with a latency of approximately 200 ns. To avoid

this latency overhead, I add a single twisted-pair (LEMO) cable between GNN-ETM and ICN-ETM. This transmission interfaces directly with the high-speed-capable pins on the FPGA, avoiding error correction, synchronisation, and other physical-layer overhead present in the GT connections. On the GDL, trigger bits are received from all L1 Trigger subsystems. For simplicity, only ICN-ETM and GNN-ETM are shown in the figure. In general, trigger bits pass through three submodules:

First, the input trigger delay submodule checks the connection to the GDL for liveness and measures the subtrigger latency via the `active` signal. Variable-length shift registers are used to synchronise all incoming subtriggers based on the measured delay. The GDL implements rate counters for the `active` signal and all trigger bits after this stage, measuring the raw input trigger rate and storing it in the EPICS archiver database. trigger bits included in the GDL are monitored via the Input Trigger Delay Process Variables (PVs).

Second, an optional veto (bitwise `AND` with the inverted veto signal) is applied to the input trigger signals and again recorded as Final Trigger Decision PVs. Two vetoes used for the GNN-ETM trigger signals are the injection veto and the Bhabha veto. The injection veto suppresses beam-induced background. The Bhabha veto suppresses the high-rate Bhabha scattering process. Without applying these vetoes, both would dominate the resulting trigger rates.

Third, a prescale and mask value is applied to all trigger bits. This effectively enables the run operators to disable single bits (masking) or to reduce the trigger rate of these bits by triggering only on the nth occurrence (where n is the prescale factor). The output of this stage is recorded as Prescale and Mask PVs.

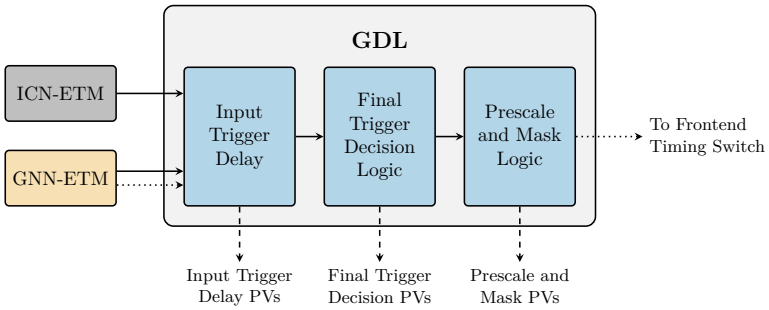


Figure 6.23: Interfaces between GNN-ETM, ICN-ETM and GDL. Interfaces via GT are shown as solid arrows. Interfaces via twisted-pair cables are depicted as dotted arrows. Slow control interfaces are depicted as dashed arrows. Taken from ref. [Neu26d].

Due to the large number of subdetectors and subsystems in the L1 Trigger system, only five trigger bits from GNN-ETM can be included simultaneously in the monitoring pipeline. The GDL firmware is modified to include the trigger bits from the GNN-ETM as inputs (see also appendix B). Table 6.6 gives the configuration of GNN-ETM trigger bits on the GDL and their respective vetoes in June 2026. The selection reflects the constraint of five available monitoring slots and covers a representative subset of the complete GNN-ETM trigger output. All signal and cluster-multiplicity bits are vetoed against both Bhabha events and injection background, whereas `gnn_lml7` is vetoed only against injection background as the selection of the low-multiplicity bits is restrictive by construction (see also section 6.2). Most of the trigger bits are transmitted via the GT interface, as the twisted-pair cable supports only a single trigger bit. The trigger bit sent over a twisted-pair cable is reconfigurable at runtime via the slow control interface (see also section 6.5).

Table 6.6: Exemplary GDL configuration for GNN-ETM trigger bits in June 2026. GT equals transmission via gigabit transceiver. LEMO equals transmission via twisted-pair cable.

trigger bit	Input Rate Id	Final Rate Id	Interface	Veto
gnn_cpt_c2_100mev	140	40	GT	Bhabha & Injection
gnn_sig_c2_100mev	82	58	LEMO	Bhabha & Injection
gnn_sig_c3_100mev	141	101	GT	Bhabha & Injection
gnn_sig_c4_100mev	142	107	GT	Bhabha & Injection
gnn_lml7	144	111	GT	Injection

In this work, GNN-ETM trigger bits are monitored in the GDL, but do not actively contribute to the trigger decision in the Belle II L1 trigger system. In the following, GNN-ETM will be evaluated in runs. A run is defined as a data-taking period during which the experiment and the accelerator configuration remain constant. In the following, I differentiate between two run types. Cosmic runs are data-taking periods without beam, whereas beam runs are data-taking periods in which the beams collide in the interaction point of the Belle II Experiment.

6.8.2 Latency Measurement

I derive the end-to-end latency and the latency requirement for GNN-ETM in a beam run. The system latency is measured by observing the histogram of the rising-edge clock counter implemented on the GDL, recorded over the full run. The observed input delay is denoted by $\hat{t}$. The rising-edge clock counter continuously monitors each trigger bit and stores the delay value with a resolution of 32 ns for both the gigabit transceiver and the twisted-pair

cable. For the analysis, a histogram of arrival times is recorded over a full run using a dedicated clock counter module on the GDL. The upper acceptable input delay at the GDL has been defined as 20 system clock cycles, or 640 ns, based on experimental evaluation during data acquisition (DAQ) stress tests. The input delay is measured against an arbitrarily chosen reference time on the GDL, which depends on the clock distribution architecture at Belle II.

To relate a connection between $\hat{t}$ and the actual GNN-ETM latency $\hat{t}_{\text{gnn}}$, I apply the following offsets: First, I apply the programmable delay offset t_{FAM} from the Frontend Analog Module to remove the effects of different run configurations. Second, I apply an offset based on the difference between the simulated cycle-accurate latency t_{sim} and the 95 % quantile $\hat{t}_{0.95}$ measured via the twisted-pair cable:

$$\hat{t}_{\text{gnn}} = \hat{t} + t_{\text{FAM}} + t_{\text{sim}} - \hat{t}_{0.95} \tag{6.15}$$

Figure 6.24 shows the adjusted latency measurement with $t_{\text{sim}} = 1053\,\text{ns}$, and $\hat{t}_{0.95} = 480\,\text{ns}$. The latency distribution of the twisted-pair cable is wider than the latency distribution of the transmission via gigabit transreceiver. This difference is likely due to the missing timing constraints are on the inter-FPGA datapath. The widened latency distribution likely results from metastability at the clock domain crossing, caused by the lack of a fixed phase relationship between the two FPGA clocks. For the configuration $t_{\text{FAM}} = -146\,\text{ns}$, the latency margin on GNN-ETM is 160 ns for trigger bits transmitted via twisted-pair cable. Without this configuration delay offset, the latency margin shrinks to 14 ns. In both configurations, the latency of the GNN-ETM is too high to transmit trigger bits over gigabit transceivers, due to the overhead of the physical communication layer.

Three latency bounds can therefore be derived from experimental measurements:

1. In the current configuration, a latency bound of 1067 ns is derived.
2. When the programmable delay offset is applied at the Frontend Analog Module, the latency budget increases to 1213 ns.
3. Additionally swapping GNN-ETM and ICN-ETM increases the latency budget further up to 1367 ns.

To conclude, GNN-ETM is ready to partake in active trigger decisions of the Belle II L1 trigger system with a single, runtime-reconfigurable trigger bit via twisted-pair cable.

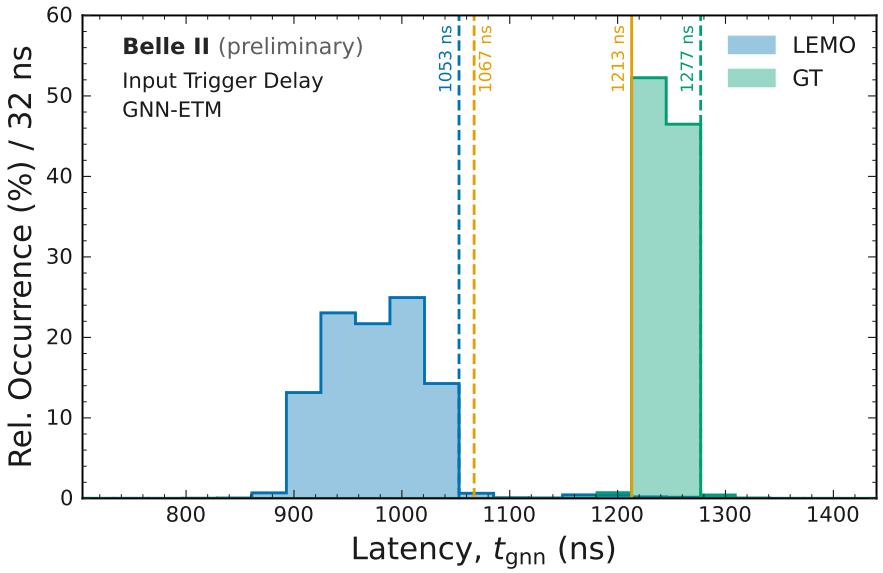


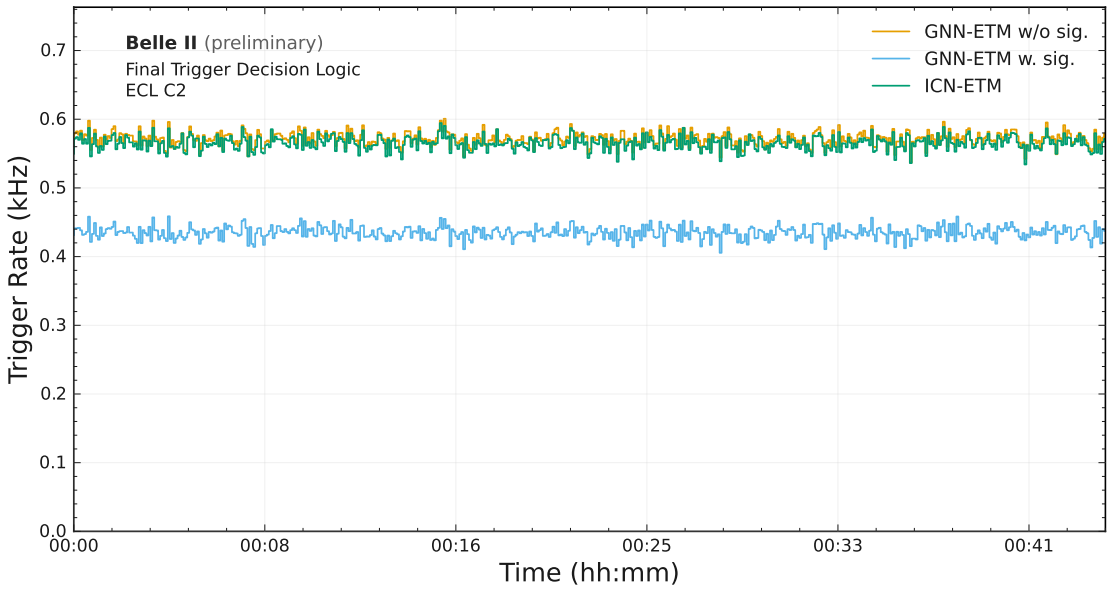
Figure 6.24: Relative occurrence of the GNN-ETM latency $\hat{t}_{\text{gnn}}$ of a trigger bit from GNN-ETM, measured at the GDL. GT equals transmission via gigabit transceiver. LEMO equals transmission via twisted-pair cable. Blue and green dashed lines denote the 95 % quantile for the respective distribution. The orange line describes the latency requirement for partaking in the active trigger decision with $t_{\text{FAM}} = -146\,\text{ns}$. The dashed orange line describes the latency requirement for partaking in the active trigger decision with $t_{\text{FAM}} = 0\,\text{ns}$. Taken from ref. [Neu26d].

6.8.3 Trigger Rate Monitoring

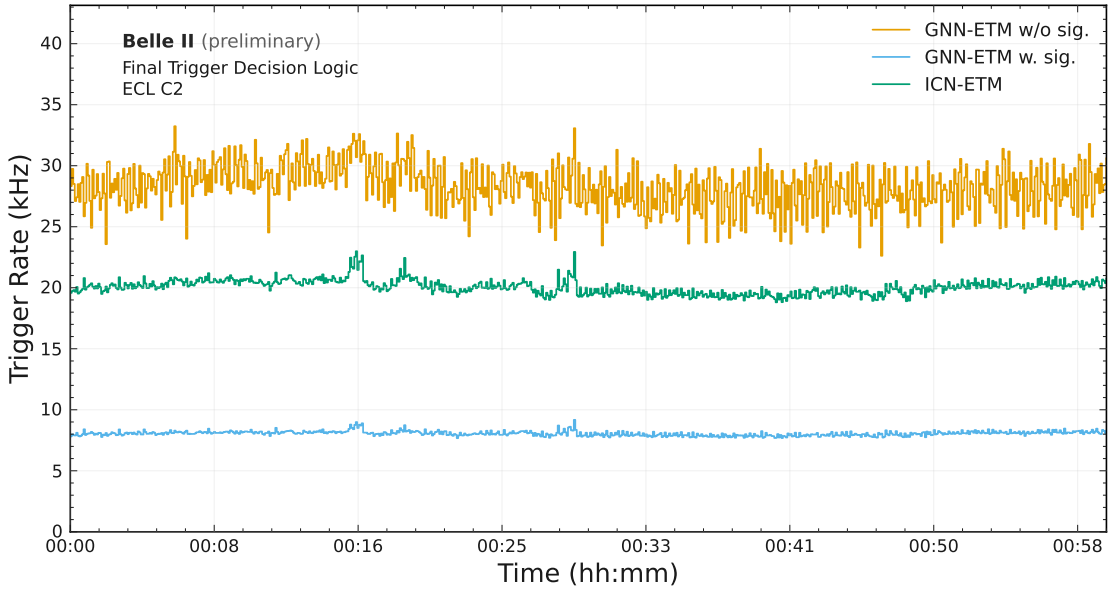
In the following, I compare the trigger rates for the C2 trigger bit on the existing ICN-ETM and GNN-ETM, using this representative trigger bit to demonstrate trigger rate monitoring. The C2 trigger bit is a Boolean decision variable which is true if at least two clusters are detected in the ECL inner region in the 250 ns observation window of the ECL L1 Trigger system (see also section 6.2). In both systems, a per-cluster energy cut of 100 MeV is applied. The C2 trigger bit is choosen due to its straightforward implementation, even though it does not allow any conclusion to be drawn about the physics process which produced the clusters.

Figure 6.25 shows a comparison of trigger rates between the ICN-ETM and the GNN-ETM for the C2 trigger bit for technical demonstration. I select two representative runs from the Belle II operation between May and June 2026 to demonstrate the functionality of the trigger rate monitoring. Figure 6.25a shows a cosmic run without beam collisions in June 2026. In cosmic runs, neither beam background nor no photons are detectable. Instead, the rate is dominated by downgoing muons that create energy deposition patterns that where not part of the GNN training. Figure 6.25b shows a physics run with beam collisions in May 2026. The rates are based on the Final Trigger Monitor PVs on the GDL after applying both the Bhabha and injection vetoes. As a baseline, I depict the trigger rate of the existing ICN-ETM C2 trigger bit. For the GNN-ETM, I show two versions: First, the C2 trigger bit is shown based on the clusters selected by the condensation-point selection algorithm without consideration of the signal classifier per-cluster output [Neu26a]. This trigger rate is denoted as GNN-ETM w/o sig. Second, the same C2 trigger bit is shown, but the signal classifier is applied to each cluster. As a result, background clusters are masked, and the total number of clusters per detector snapshot decreases, so the threshold of at least two clusters is reached in fewer cases. This trigger rate is denoted as GNN-ETM w. sig.

In Figure 6.25a, I observe that the C2 trigger rates of the ICN-ETM and the GNN-ETM without a signal classifier are almost identical. Applying the signal classifier on GNN-ETM reduces the C2 trigger rate by approx. 100 Hz. In Figure 6.25b, I observe a higher C2 trigger rate for the GNN-ETM w/o sig. in comparison to the ICN-ETM. A potential cause of this increased rate is the ability of GNN-ETM to split energy depositions into multiple clusters. Another plausible cause is the characteristic of the ICN-ETM, to shift the position of a cluster towards the forward endcap due to the way TCs are defined in the inhomogeneous endcap region [Neu26a]. Thus, the ICN-ETM is more likely to have only one cluster in the ECL inner region ($1 < \theta_{\mathrm{id}} < 16$), as required by the C2 trigger bit, which leads to a lower trigger rate



(a) Cosmic run



(b) Beam run

Figure 6.25: Comparison of C2 trigger rates between ICN-ETM and GNN-ETM based on monitoring PVs on the GDL. GNN-ETM trigger rates are shown both with the signal classifier (w. sig.) and without the signal classifier (w/o sig.). Taken from ref. [Neu26d].

than the GNN-ETM. After applying the signal classifier, the trigger rate of GNN-ETM drops significantly below the ICN-ETM rate.

In general, both online measurements of the GNN-ETM and the ICN-ETM confirm the trends presented in ref. [Neu26a]:

1. For cosmic runs without beam background, the cluster-finding performance of ICN-ETM and GNN-ETM w/o sig. is almost identical.
2. For cosmic runs without beam background, the GNN-ETM signal classifier can significantly reduce the trigger rate.
3. For beam runs, a greater difference in trigger rates is expected and observed between GNN-ETM and ICN-ETM.

Nevertheless, a more thorough analysis is required to make quantitative statements on the two systems. By making online monitoring available, this work lays the groundwork for a quantitative comparison of the two modules in the Belle II ECL L1 Trigger system.

6.9 Discussion

This chapter has presented the GNN-ETM and demonstrated the deployment of a dynamic PCN, CaloClusterNet, onto FPGA in the Belle II L1 Trigger system. The implementation targets the UT 4 with an AMD Ultrascale XCVU190 and supports up to 32 non-zero inputs. The system architecture comprises a preprocessing stage, a dataflow accelerator, and a post-processing stage, and has been validated in C-level simulation, RTL-level simulation, and on hardware. The GNN-ETM has been commissioned and has been operating in the Belle II L1 Trigger system since the end of 2024. Since the beginning of 2026, it has fulfilled the latency requirement of 1067 ns for partaking in an active trigger decision, whilst sustaining a throughput of 8 million detector snapshots per second. Monitoring of trigger rates and integration into the Belle2Link readout system enable continuous monitoring and debugging. During operation, the system has performed stably and as expected, with no anomalies observed in either the trigger output or the monitoring readout. The GNN-ETM constitutes the first deployment of a dynamic PCN operating in a hardware trigger of a particle physics experiment [Neu26a], demonstrating that dynamic PCN inference is feasible within the strict real-time constraints of a hardware trigger. Compared to the ICN-ETM, the GNN-ETM removes the fixed six-cluster limit inherent to the template-matching approach and introduces

a signal classifier that suppresses beam-induced background. A notable finding from the design exploration is that disabling DSP resources in the dataflow accelerator reduces both latency and routing congestion: LUT-based multipliers at low fixed-point precision require shallower pipelines and impose fewer placement constraints than DSP blocks, improving routeability despite an increase in LUT utilisation.

For future upgrades, the latency budget can be increased to 1367 ns by modifying the Belle II L1 Trigger architecture. The 14 ns margin of the final design leaves no headroom for retraining the model without risking a latency violation; adapting the trigger system architecture is therefore recommended before any model update is attempted. Furthermore, the current system supports only a single trigger bit transmitted via twisted-pair cable for participation in active trigger decisions; expanding this to multiple bits is essential for a full replacement of the ICN-ETM in future upgrades. The scalability to input sizes beyond 32 non-zero inputs likewise remains a challenge on the current hardware platform. Taken together, these results establish that dynamic PCN inference is viable in a production hardware trigger and identify trigger system integration, rather than the neural network itself, as the primary constraint on future performance. The following chapter explores heterogeneous SoCs as an alternative platform to address the remaining scalability limitations.

Chapter 7

Dynamic Point Cloud Networks on Heterogeneous Systems-on-Chips

The work presented in this chapter has been presented in refs. [Neu25b, Neu26b].

In chapter 6, the deployment of a PCN-based clustering algorithm within the Belle II L1 Trigger was demonstrated. The dataflow accelerator of the GNN-ETM is dimensioned for up to 32 input points per inference, which is sufficient under current Belle II operating conditions. The planned upgrades during long shutdown 2, however, motivate a study of how graph-based PCNs scale to larger input cardinalities. To this end, this chapter presents a case study on the AMD Versal heterogeneous SoC platform (see section 2.3.3), targeting the CaloClusterNet model together with the CPS algorithm. I co-optimize the deployment across the FPGA and AIE fabric through operator fusion, graph partitioning, explicit kernel placement, and compare the resulting throughput and latency to a GPU baseline. The chapter closes with an end-to-end hardware demonstrator that integrates the dataflow accelerator with a visualization pipeline and an interactive web interface, also serving as an outreach platform.

The remainder of this chapter is structured as follows. Section 7.1 introduces the underlying PCN model and derives the design requirements from the planned Belle II upgrade during long shutdown 2. Section 7.2 details the deployment and optimization strategies with a focus on the co-optimization across the FPGA and AIE fabric. Section 7.3 describes the hardware architecture of the demonstrator and the experimental setup. Section 7.4 presents the performance evaluation. Section 7.5 summarizes and discussed the findings of this case study.

7.1 Algorithm and Problem Statement

The development of the algorithm and the training of the neural network have been thoroughly studied in ref. [182]. In the following, the model is summarised to provide the context needed to derive the system requirements.

For the planned Belle II upgrade of both accelerator, interaction region, and detector, internally denoted as Long Shutdown 2, different configurations for upgrading the ECL detector are under investigation. Currently, all 8736 crystals are compressed into 576 TCs as a preprocessing step to reduce the computational load on the L1 Trigger system (see also section 2.1.5). This preprocessing limits the performance of clustering algorithms in two ways. First, the spatial resolution is reduced because only the centre of each TC is available as a candidate cluster position. Thus, two clusters in close proximity cannot be separated if both deposit energy in the same TC. Second, the analog summation accumulates noise across all contributing crystals, which forces the energy threshold to be set to 100 MeV to suppress noisy TC channels. As a consequence, low-energetic particles may not be detected.

The proposed upgrade removes this bottleneck by exposing individual crystal information to the trigger, which in turn increases the number of inputs to the CaloClusterNet model. This poses a significant challenge for deployment, as the model's computational complexity scales as $O(n^2)$ with the number of inputs n. To estimate the expected input distribution after Long Shutdown 2, a study of three data-reduction methods was conducted in ref. [182]. The results of this study are summarised in figure 7.1. Two of these data-reduction methods apply flat-energy cuts to all crystals at 6.25 MeV and 20 MeV. The third data-reduction method further reduced the flat-energy cut to 5 MeV without increasing the number of active crystals by introducing a tower and distance cut. Towers represent regions of interest around higher-energy crystals. Depending on the selected energy threshold and distance cuts, the number of non-zero crystals per event varies considerably. Based on this estimate, at least 128 of 8736 sparse inputs must be processed per event without reducing overall throughput. This is a factor of four more inputs than the current deployment on the GNN-ETM, which processes up to 32 non-zero sparse inputs out of 576 per event (see also chapter 6). Owing to the quadratic scaling of computational complexity, this results in a 16-fold increase in computational load.

To address this challenge, a case study is developed to deploy a CaloClusterNet model under the operating conditions expected for Belle II after Long Shutdown 2, using the currently operating GNN-ETM as the reference baseline. Figure 7.2 shows the network topology of the adapted CaloClusterNet model for this case study. The architecture

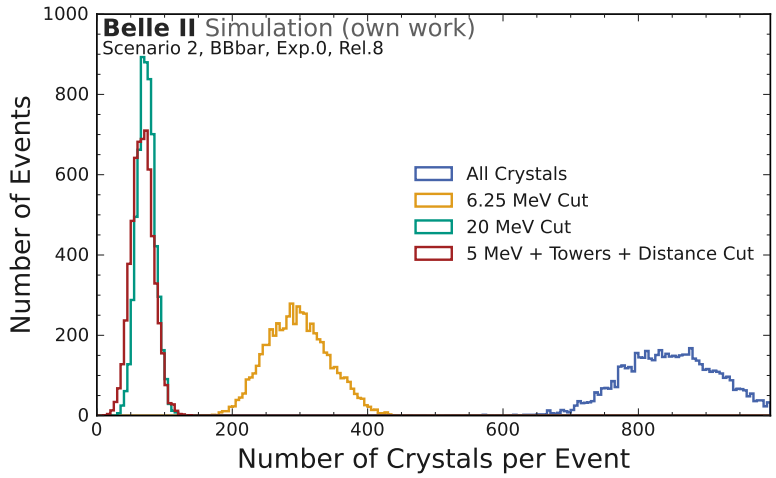


Figure 7.1: Expected number of crystals per trigger window for three data reduction methods based on beam background simulations. Taken from ref. [182].

was optimised in ref. [182] for performance evaluation on single crystals as inputs, in contrast to the existing version, which relies solely on TCs. More details are given in appendix A.3.

For the deployment of this adapted CaloClusterNet model, the following requirements are derived:

Requirement 7.1. The deployment shall process at least 128 non-zero crystal inputs per detector snapshot, in line with the estimation derived from figure 7.1. This is a factor of four more inputs than the GNN-ETM baseline accepts.

Requirement 7.2. The end-to-end inference latency shall not exceed 10 µs. This is anticipated to relax the current hard latency deadline of 5 µs for the L1 Trigger pipeline by extending the DAQ readout buffers during Long Shutdown 2.

Requirement 7.3. The deployment shall sustain a throughput of $R_{\text{th}} = 8.00$ million detector snapshots per second, matching the GNN-ETM baseline. The use of multiple platforms in spatial parallelism is permitted to meet this throughput target.

For the scope of this case study, the platform choice is left open. The AMD Versal is selected as the deployment platform. It is a commercially available heterogeneous SoC (see also section 2.3.3). In addition to the FPGA fabric, it integrates a CGRA partition with AIE engines. These engines offer an improved performance-to-area ratio for compute-intensive tasks [92].

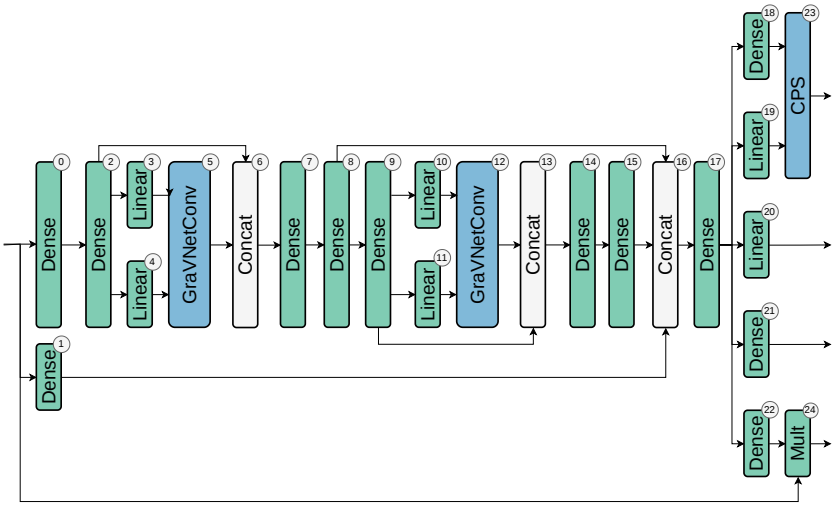


Figure 7.2: CaloClusterNet architecture developed in ref. [182] for an upgrade of the Belle II ECL detector after Long Shutdown 2.

7.2 Deployment and System Design

Deploying CaloClusterNet onto the AMD Versal, a heterogeneous SoC, requires translating the algorithmic description in section 7.1 into a hardware implementation that satisfies the corresponding requirements. The toolflow used for this translation is outlined in section 7.2.1. The essential transformation steps for the deployment are then examined in detail: the partitioning of the network in section 7.2.3 and the mapping of the neural network in section 7.2.4. Subsequently, specific optimisations, namely operator fusion, spatial parallelism, and kernel-level optimisations, are discussed in sections 7.2.2, 7.2.5 and 7.2.6, although they are applied at different points in the toolflow.

7.2.1 Toolflow Methodology

To support deployment of CaloClusterNet onto heterogeneous SoCs such as AMD Versal, I develop a Python-based semi-automated design flow. The design flow requires two inputs: a pretrained, quantised neural network model and a set of hardware requirements, including the target throughput and target platform. The network is assumed to have been trained and quantised with QKeras [183]. As output, the flow emits a bitstream for the FPGA partition and a binary container for the AIE partition on AMD Versal platforms.

An overview of the deployment flow is shown in figure 7.3, separated into a series of stages Ⓐ–Ⓗ. The following gives a brief overview of the end-to-end toolflow:

The flow starts with the operator fusion stage, simplifying the dataflow graph of the neural network where applicable Ⓐ.

Next, the operators in the model are partitioned onto their deployment targets Ⓑ. For this purpose, system requirements are combined with performance measurements from two module libraries: an HLS module library targeting the FPGA partition and an AIE kernel library targeting the AIE partition. The result of the partitioning step is a set of subgraphs, each modelling the dataflow of one partition of the network.

Subsequently, the subgraphs are mapped onto their hardware targets Ⓒ. For each neural network layer, a PE is chosen based on the architecture templates in the DeePloy library (see chapter 4).

For each partition, spatial parallelisation is applied to the mapped design Ⓓ. For this step, the smallest parallelisation factor that still satisfies the system's throughput requirements is chosen.

As the final optimisation stage, kernel-level optimisations are applied to AIE kernels Ⓔ. This step fine-tunes the selection of AIE kernel templates in case multiple versions of the same PE are available.

The semi-automated toolflow automatically generates weight and configuration files for all individual PEs for both AMD Vitis HLS and the Chess Compiler Ⓕ. However, the user is responsible for mapping the network topology into the corresponding C++ source files.

For system integration, the generated Verilog code from the HLS toolchain is combined with the binaries generated by the Chess compiler via AMD Vitis Ⓖ. For implementation on the target platform, the v++ tool is used to export a design for AMD Vivado for synthesis, implementation, and packaging Ⓗ.

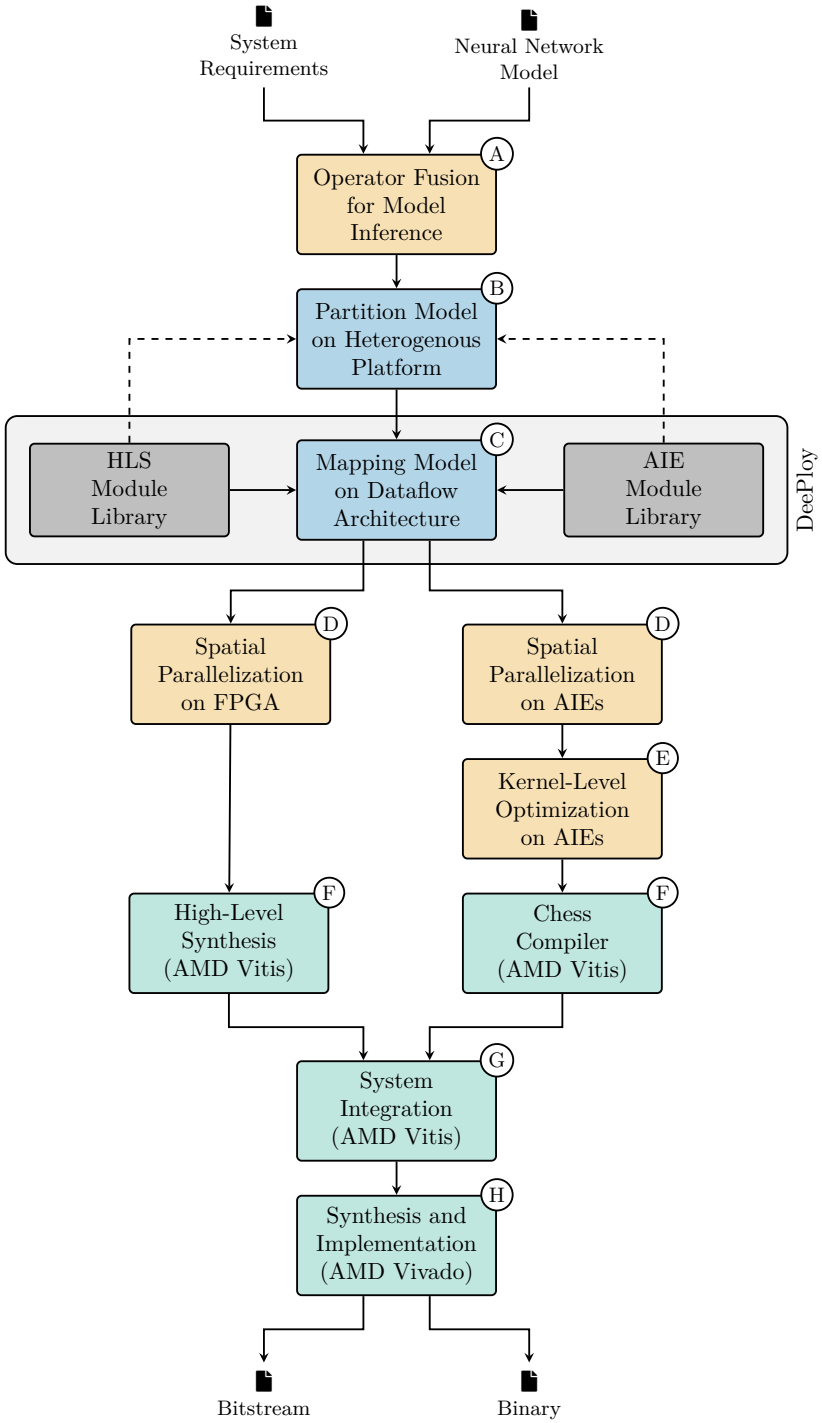


Figure 7.3: Overview of the developed deployment flow. Circled letters identify the stages in the flow. Key transformation stages are blue. Optimisation stages are orange. The existing toolchains used for deployment are green. Adapted from ref. [Neu26b].

7.2.2 Operator Fusion

Two graph-level transformations are applied before mapping. First, Linear layers and their subsequent ReLU activations are fused into a single Dense operator. Second, parallel Dense operators with the same predecessor are merged into a single Dense operator whose output features are the concatenation of the individual operators' outputs. These optimisations simplify the dataflow graph topology by removing the multicast on the preceding operator. The simplified topology is particularly critical, as no more than eight AIE memory buffers are connected to each tile on the first generation of AMD Versal. Introducing even a single multicast requires four memory buffers due to double buffering, leaving insufficient buffers for the remaining operators during tile placement.

7.2.3 Partitioning

Next, we assign each operator to one of two target platforms: FPGA logic or AIE tiles. I employ a greedy scheme that prioritises the AIE platform, as AIE tiles offer better performance per area. The single-pass greedy partitioning algorithm is described in algorithm 7.1.

I adapt the standard formulation of resource binding in HLS [184]: Let $\mathcal{R}$ denote a set of resource types of the target architecture, partitioned into $\mathcal{R}_{\text{fpga}}$ and $\mathcal{R}_{\text{aie}}$. For each resource type $r \in \mathcal{R}$, $\text{ops}(r) \subseteq \mathcal{V}$ denotes the set of operators implementable on r. An operator $v \in \mathcal{V}$ is *legal* on the AIEs if there exists a resource type $r \in \mathcal{R}_{\text{aie}}$ with $v \in \text{ops}(r)$. The subset $\mathcal{V}_{\text{io}} \subseteq \mathcal{V}$ denotes the input and output operators that interface with DDR-RAM memory and are therefore constrained to FPGA logic.

In practice, an operator is *legal* on a given partition if the respective library (see chapter 4) implements it. All operators with regular, statically scheduled data access patterns are assigned to the AIE platform, including layers such as Linear, Dense, ReLU, or Concat, but excluding PCN-specific layers which require data-dependent memory access. All AIE kernels run fully decoupled, without global memory access, avoiding memory stalls at runtime. Based on the resulting subsets, subgraphs are defined by retaining the original edges of the dataflow graph within each subset. Unconnected subgraphs are labelled independently. The resulting partitioning is shown in figure 7.4: seven partitions are derived, of which four are implemented on FPGA and three on AIEs.

Algorithm 7.1 Single-Pass Greedy Partitioning

procedure PARTITION($\mathcal{V}$)
  $\mathcal{V}_{\text{fpga}} \leftarrow \emptyset$
  $\mathcal{V}_{\text{aie}} \leftarrow \emptyset$
  for $v \in \mathcal{V}$ **do**
    if $v \in \mathcal{V}_{\text{io}} \vee \nexists r \in \mathcal{R}_{\text{aie}} : v \in \text{ops}(r)$ **then**
      $\mathcal{V}_{\text{fpga}} \leftarrow \mathcal{V}_{\text{fpga}} \cup \{v\}$
    else
      $\mathcal{V}_{\text{aie}} \leftarrow \mathcal{V}_{\text{aie}} \cup \{v\}$
    end if
  end for
  return $\mathcal{V}_{\text{fpga}}$, $\mathcal{V}_{\text{aie}}$
end procedure

7.2.4 Mapping of the Neural Network

After partitioning, each operator is mapped to a corresponding architecture template through pattern matching. For deploying the pretrained, quantised CaloClusterNet (see section 7.1) on AMD Versal, I use the DeePloy (see chapter 4) design methodology. For FPGA operators, HLS templates from section 4.5 are used, which support the irregular, data-dependent access patterns required by graph convolutions (*GravNetConv*) and the Condensation Point Selection (*CPS*) algorithm. For AIE operators, C++ templates from section 4.5 are used, providing reusable templates for Dense, Linear, Concat, and ReLU operators. Each AIE template encapsulates a single operator as a self-contained kernel.

One important distinction, compared to the HLS-only mapping flow used in section 6.4.3, is the legalization of the dataflow: AIE kernels in the architecture template library require a specific memory tiling for both the input buffers and the weight buffers. Thus, three types of additional PEs must be inserted to yield a legal dataflow architecture.

First, on the AIE partitions, a retiling kernel is required between two successive dense PEs. The reason for this lies in the architecture of the first generation of the AMD Versals used for this work. To maximise hardware efficiency on matrix multiplication, the input buffer must be tiled 8×4, and the weight buffers must be tiled 4×8. However, the generated output buffer employs a 4×4 tiling. A retiling PE is therefore inserted to convert the 4×4 tiled memory buffer to an 8×4 tiled memory buffer.

Second, a similar retiling PE is required between the FPGA and AIE partitions. DeePloy HLS architecture templates expect row-major ordering of the input buffers. Therefore, on the

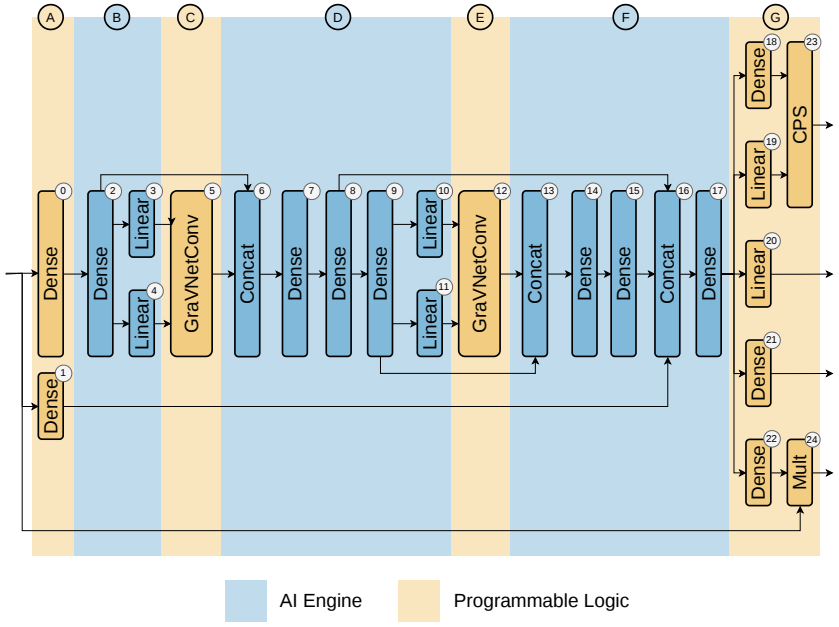


Figure 7.4: Partitioning of the Long Shutdown 2 CaloClusterNet shown in figure 7.2.

transition from FPGA to AIE, a tiling PE is inserted, and on the transition from AIE to FPGA, a detiling PE is inserted.

Third, additional PEs interfacing between the FPGA and AIE partitions must be inserted. To interface the AIE fabric via PLIOs, an AXI-Stream with an explicit `tlast` side channel must be used. Otherwise, undefined behaviour might occur on hardware, even if the transaction-level simulation in AMD Vitis works as expected. In addition, streaming interfaces must use the `ap_axiu<bitwidth, 0, 0, 1>` datatype with a bitwidth of 32 bit, 64 bit, or 128 bit. This differs from the `std::array<...>` type which DeePloy uses internally. An interface PE is inserted to handle the transformation of data types for correct compilation.

7.2.5 Spatial Parallelization

After mapping, each AIE partition is optimised through spatial parallelisation. Neural network layers that do not perform feature aggregation across neighbours are fully spatially separable, meaning they can process independent spatial regions in parallel. The operator fusion step further improves separability by reducing each AIE partition to a linear chain of operators, eliminating data dependencies. For each partition, a spatial parallelization

factor $p \in \{2^k \mid k \in \mathbb{N}_0,\ 2^k \leq p_{\max}\}$ is selected, which replicates the PEs in the partition up to $p_{\max}$ times. On the AIE, resource utilisation scales linearly with p; on the FPGA, growth is superlinear due to additional routing and buffering overhead. A search over powers of two up to $p_{\max}$ is applied to find the smallest p that satisfies the target throughput R_{thr}, minimising resource utilisation. If no $p \leq p_{\max}$ satisfies R_{thr}, the partition is reported as infeasible under the given constraints.

Algorithm 7.2 formalizes this search. For each partition subgraph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i) \in \mathcal{G}$, the parallelisation factor p_v of every vertex $v \in \mathcal{V}_i$ is doubled until the throughput get_throughput(v, p_v) meets the requirement or the upper bound is reached. Here, get_throughput(v, p_v) denotes a lookup in a precomputed database of experimentally evaluated throughput values for vertex v at parallelisation factor p_v. To preserve uniform parallelisation across the design partition, the subgraph-wide factor is set to the maximum of all per-vertex factors in $\mathcal{V}_i$.

Algorithm 7.2 Spatial Parallelism Derivation

procedure DERIVEPARALLELISM($\mathcal{G}$, R_{thr}, $p_{\max}$)
  for $G_i \in \mathcal{G}$ **do**
    $p_i \leftarrow 1$
    for $v \in \mathcal{V}_i$ **do**
      $p_v \leftarrow p_i$
      while get_throughput$(v, p_v) < R_{\text{thr}}$ and $p_v < p_{\max}$ **do**
        $p_v \leftarrow 2 \cdot p_v$
      end while
      $p_i \leftarrow \max(p_i, p_v)$
    end for
  end for
  return $\{p_i\}$
end procedure

7.2.6 Kernel-Level Optimizations

Finally, individual AIE kernels are optimised to meet the strict timing constraints of our application. For example, to sustain a throughput of 1 MHz, each kernel must complete within 1 µs. At this scale, loop pipelining overhead imposed by the Chess compiler becomes significant relative to the total kernel runtime. The AMD Vitis AIE library [185] amortises pipelining setup cost over many iterations, which is inefficient for the small matrix dimensions used here. The original code is shown in listing 7.1, my modification is shown in listing 7.2. I

replace loop pipelining with loop flattening via `chess_flatten_loop` annotations, trading larger program memory for better performance.

```
for (unsigned i = 0; i < (M/M_API); i+=2)
chess_loop_range(2,)
{
    ...
    for (unsigned j = 0; j < (N/N_API); j+=2)
    chess_prepare_for_pipelining
    chess_loop_range(2,)
    {
        ...
    }
}
```

Listing 7.1: Excerpt from matrix multiplication application example provided in ref. [185] for AIE kernels.

```
for (unsigned i = 0; i < (M/M_API); i+=2)
chess_flatten_loop
{
    ...
    for (unsigned j = 0; j < (N/N_API); j+=2)
    chess_flatten_loop
    {
        ...
    }
}
```

Listing 7.2: My modification to the original code from listing 7.1.

7.3 Implementation

This chapter covers the end-to-end implementation of the dataflow accelerator on the heterogeneous AMD Versal platform, based on the deployment methodology discussed in section 7.2. For the implementation, a first-generation AMD Versal platform, namely the AMD Versal VCK190, is chosen, as the DeePloy template library targets this architecture (see also section 4.5). In general, the methodology applies to all generations of AMD Versal devices. Optimisations that are specific to this generation of the AMD Versal architecture are described as such. Section 7.3.1 covers the description of the system architecture on the heterogeneous SoC. Section 7.3.2 explains the experimental setup for the performance analysis described in section 7.4.

7.3.1 System Architecture

I implement an end-to-end demonstrator on the AMD Versal VCK190, shown in figure 7.5. The board features the AMD Versal AI Core XCVC1902 with three partitions relevant for this work: an Arm Cortex-A72 CPU, an FPGA, and an AIE array. In total, three memory modules are installed on the board: one 8 GByte DDR-RAM4 module connected to the CPU (not shown in the figure), and two 4 GByte LPDDR4 modules connected to the FPGA partition. The two LPDDR4 modules are exposed to the programmable logic as two independent Advanced eXtensible Interface (AXI)-accessible memory banks, shown in the figure as DDR-RAM `Bank` 0 and `Bank` 1. The system architecture is shown together with the implemented CaloClusterNet model from figure 7.2, which has been deployed using the methodology described in section 7.2. Specifically, it shows the partitioned version (see figure 7.4) with seven compute kernels (`A` – `G`). Kernels `A`, `C`, `E`, and `G` are mappsed to the FPGA, while kernels `B`, `D`, and `F` are mapped to the AIE array, forming a dataflow pipeline that processes inference requests without CPU intervention. Input data is read from DDR-RAM `Bank` 0 by the `Load` kernel on the FPGA, which feeds the first compute kernel of the accelerator pipeline. Results are written back to DDR-RAM `Bank` 1 by the `Store` kernel. Both `Load` and `Store` communicate with the DDR-RAM via AXI DMA.

The control flow of the accelerator is handled by the CPU, which runs PetaLinux and interfaces with the accelerator via Xilinx Runtime (XRT). I develop a C++ static library which directly interfaces with the accelerator chain via the XRT API. This HAL covers: First, managing the kernel lifecycle (loading the `xclbin` and starting and stopping individual kernels). Second, moving input and output data between host memory and the LPDDR4 banks. Third, configuring runtime parameters such as the isolation and beta thresholds of the condensation point selection algorithm, which are set once at startup. A Python wrapper exposes the HAL for better usability. Listing 7.3 shows the Python interface for running inference on the end-to-end demonstrator. `setup()` loads the `xclbin` onto the device, configuring the programmable logic and the AIE array. `load()` resets the accelerator and copies data from the NumPy array `triggercells` to DDR-RAM `Bank` 0. `run()` starts the execution for all partitions. After completion, data is read back from DDR-RAM `Bank` 1 and copied into the NumPy arrays `clusters` and `flags`.

```
from accelerator import Accelerator
with Accelerator(path) as acc:
    acc.setup()
    acc.load(triggercells)
    clusters, flags = acc.run()
```

Listing 7.3: Python code snippet for running inference on the end-to-end demonstrator.

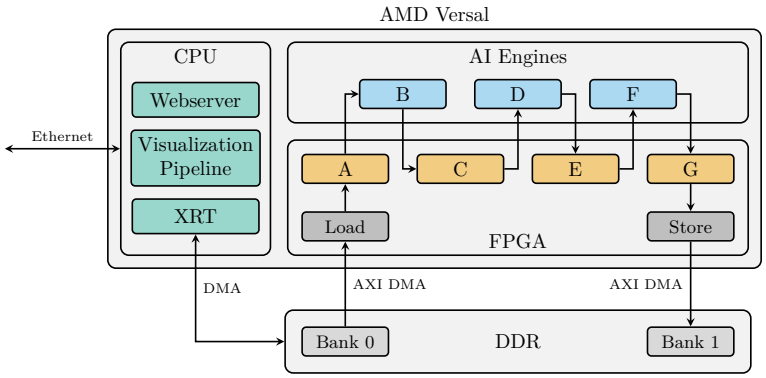


Figure 7.5: Overview of the system architecture on the AMD Versal VCK190. Partitions are implemented as individual kernels A to G as shown in figure 7.4. Adapted from ref. [Neu26b].

7.3.2 Experimental Setup

For the performance evaluation in section 7.4, I implement eight versions of the accelerator on the AMD Versal VCK190. AMD Vitis 2024.2 and AMD Vivado 2024.2 are used throughout these experiments. To reach timing closure, the `Flow_PerfOptimized_High` synthesis strategy and `Performance_ExplorePostRoutePhysOpt` implementation strategy are used in AMD Vivado. Functional correctness is validated in software using QKeras [183], in hardware emulation, and on hardware, with bit-accurate agreement across all three.

Table 7.1 gives an overview of the configuration parameters for all eight implementations. The following parameters are varied: First, five implementations use the FPGA fabric, while three additional implementations use both the FPGA and AIE fabrics. Second, to explore the design space, the fixed-point bitwidths of the PCNs are varied. On FPGA, all neural network layers use either 8 bit or 16 bit quantization. The FPGA & AIE versions use 16 bit precision for partitions A and G, which interface with DDR memory and process raw sensor values whose dynamic range exceeds 8 bit, and 8 bit for all remaining partitions. For all hyperparameters of this network see appendix A.3. Third, to evaluate the maximum event size n, the parameter is varied in the interval $n \in \{32, 64, 128\}$. All versions support up to n inputs per inference; fewer inputs are supported through zero-padding. The smallest version with $n = 32$ is functionally identical to the GNN-ETM and serves as a baseline, despite the different target FPGA.

The FPGA & AIE versions are only implemented for $n = 128$, where the FPGA-only version fails to route (see section 7.4). To evaluate the impact of my optimisation stages from section 7.2,

				Parameters		Frequency	
Target	Precision	n	Iteration	p_{fpga}	p_{aie}	f_{fpga}	f_{aie}
FPGA	8 bit	32	–	2	–	290 MHz	–
FPGA	16 bit	32	–	2	–	260 MHz	–
FPGA	8 bit	64	–	2	–	280 MHz	–
FPGA	16 bit	64	–	2	–	249 MHz	–
FPGA	8 bit	128	–	1	–	250 MHz	–
FPGA & AIE	mixed	128	①	1	1	250 MHz	1.250 GHz
FPGA & AIE	mixed	128	②	2	4	250 MHz	1.250 GHz
FPGA & AIE	mixed	128	③	2	4	250 MHz	1.250 GHz

Table 7.1: Overview of configuration parameters for the eight evaluated implementations. The FPGA-only version with $n = 32$ corresponds to the GNN-ETM baseline.

I define three successive design iterations:

- ① serves as a baseline, implementing the partitioned network (see section 7.2.3) without further optimisations.
- ② applies operator fusion and spatial parallelization (see also sections 7.2.2 and 7.2.5).
- ③ additionally applies the kernel-level optimizations (see also section 7.2.6).

In table 7.1, I denote the parallelisation factor for all partitions implemented on FPGA and AIE as p_{fpga} and p_{aie}, respectively. The FPGA-only version with $n = 128$ and 16 bit quantisation failed to route in implementation. The same happened for the FPGA-only version with $n = 128$ and 8 bit quantisation when p_{fpga} was increased. When AIEs are used, f_{fpga} is chosen as an integer divisor of f_{aie} to simplify clock domain crossings. Asynchronous clock domain crossings incur nondeterministic latency at the domain boundaries.

7.4 Performance Analysis

This section covers the performance analysis of the eight implementations outlined in section 7.3.2. The evaluation is split into four experiments: First, the throughput and end-to-end latency of the FPGA implementations are compared with a GPU baseline. Second, the throughput and end-to-end latency of the FPGA & AIE implementations

are compared with a GPU baseline. Third, a latency ablation study identifies bottlenecks in the FPGA and AIE implementations. Fourth, system resource utilisation is analysed.

As a baseline, I deploy the CaloClusterNet from section 7.1 on an NVIDIA GPU L40S in a server with an AMD EPYC 9554 CPU. The model is compiled from PyTorch [147] using TensorRT [186], with the maximum optimisation level enabled in the `trtexec` tool. To establish a strong baseline on general-purpose compute platforms, I evaluated several state-of-the-art frameworks, including PyTorch Geometric [73], ZenTorch [187], and FastGraphCompute [188], but none outperformed the others. As a data type, I chose Brain Floating Point 16 [189]. Optimising the model with a fixed-point-quantised version did not yield better overall performance on GPU.

In the following, latency refers to the end-to-end latency of a batch-one execution, and throughput refers to the maximum throughput obtained across all evaluated batch sizes. For the GPU measurements, wall clock time generated by the TensorRT benchmark tool is used. The batch size $\mathcal{B}$ is varied in $\mathcal{B} \in \{1, 2, \ldots, 4096\}$ to determine both the batch-one latency and the maximum obtainable throughput. To minimise jitter from OS scheduling and other workloads on the non-real-time GPU host, measurements are performed on an isolated server. A series of 1000 measurements is performed for all batch sizes and the 95 % quantile is reported. Latency and throughput of the baseline GPU are shown as solid black bars in figure 7.6.

FPGA Throughput and Latency

This subsection covers the first five implementations, which use the FPGA fabric exclusively. The remaining three heterogeneous FPGA and AIE implementations are discussed in the following subsection.

I measure the throughput of the accelerator by executing the FPGA kernel with a large batch size of 16 384 to reduce the influence of the nondeterministic host system. This does not undermine the real-time capability of the system, since all network inferences are processed sequentially. The performance gain over a batch-one execution is introduced by pipelining multiple network inferences in a row.

Performance metrics are measured via profiling monitors inserted on the FPGA to record execution timelines for all kernels. The execution time is measured, including memory

transfers, using trace files provided by the XRT Runtime, to enable a fair comparison with the GPU baseline.

Figures 7.6b, 7.6d and 7.6f show the throughput measurements. The application throughput requirement of $R_{\text{th}} = 8.00$ million detector snapshots per second, indicated as a horizontal red line in the figures, is met for event sizes of 32 but not for 64 and 128. For an event size of 64, the difference between achieved and required throughput is small. This shortcoming is casued by the control-flow overhead when using the external DDR-RAM interface, as shown in ref. [Neu25b]. The general trend of lower throughput for larger event sizes is expected, as the GravNet complexity is bound by $\mathcal{O}(n^2)$. In the end-to-end evaluation, I achieve a speedup of 3.46×, 5.25×, and 2.92× compared to the GPU baseline.

I measure the latency of the accelerator with a batch size of one. Figures 7.6a, 7.6c and 7.6e show the latency measurements. In all cases, the implementations meet the latency requirement of $R_{\text{lat}} = 10\,\mu\text{s}$, which is indicated by the horizontal red line in the figures. The increase in latency for an event size of 128 points is largely caused by the reduced p_{fpga} factor. Compared with the GPU baseline, I achieve a substantial reduction in execution latency. The GPU's flat latency profile across event sizes indicates that its execution is dominated by memory transfer and kernel launch overhead rather than by compute.

FPGA & AIE Throughput and Latency

The initial design iteration ① performs worse than the FPGA baseline in both latency and throughput. After optimisation, design iteration ② achieves a throughput of 2.36 million detector snapshots per second at a latency of 7.47 µs. The best performance is achieved by design iteration ③, with a throughput of 2.94 million detector snapshots per second and a latency of 7.15 µs. Relative to the FPGA-only baseline, design iteration ③ achieves a throughput improvement of 1.53× at a latency overhead of 1.18×. In comparison to the GPU baseline, design iteration ③ achieves a throughput improvement of 7.02×.

The performance of design iteration ① is lower than the FPGA-only version due to overhead introduced by the partitioning (see section 7.2.3), notably the crossings between FPGA and AIE as well as additional retiling PEs. The throughput of the design is considerably lower than the theoretical limit of 2.00 million detector snapshots per second.

Thus, operator fusion (see section 7.2.2) and spatial parallelisation (see section 7.2.5) are introduced as optimisations for design iteration ②. Increasing p_{fpga} to 2 and p_{aie} to 4

doubles the theoretical throughput and reduces latency by reducing pipeline depth in the PEs. In measurement, the throughput improves by 3.61× over the previous design iteration, but falls short of the theoretical throughput. After investigation, I conclude that the performance loss is caused by kernel restart and looping overhead on the AIEs. Thus, further increasing p_{aie} is unlikely to significantly improve throughput.

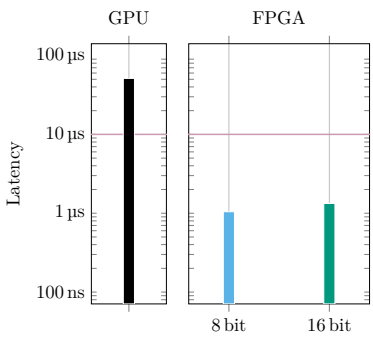
Finally, for design iteration ③, an identical tile allocation is kept, but the kernel-level optimisations are applied (see section 7.2.6). The improvement in design iteration ③ results solely from this optimisation, which reduces execution time without altering resource counts. In comparison to design iteration ②, the throughput is increased by 1.21× and the latency is further reduced by 1.05×. Design iteration ③ is the best design for $n = 128$ found in this experimental evaluation.

FPGA & AIE Latency Breakdown

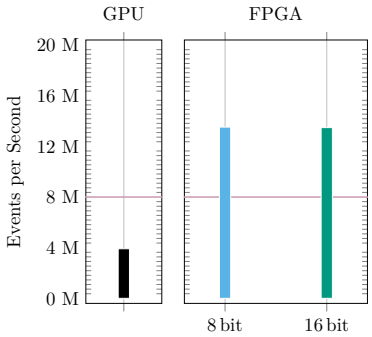
To better understand the latency contribution of each design partition, I repeat the latency measurement study from the previous subsection in hardware emulation. For this study, additional profiling IP cores are inserted on all PLIO interfaces connecting the FPGA and AIE fabric. These profiling cores increase the overall latency by 0.822 µs, 0 µs, and 0.171 µs for design iterations ①–③, respectively. This overhead originates from the routing algorithm on the AIE fabric and can vary unpredictably even with small code modifications. For this reason, I restrict the following discussion to relative values.

Figure 7.7 gives an overview of the measured latency values in hardware emulation. In design iteration ①, 36 % of the latency is spent on the FPGA side and 64 % on the AIE side. In design iteration ②, 46 % is spent on the FPGA side and 54 % on the AIE side. In design iteration ③, 47 % is spent on the FPGA side and 53 % on the AIE side.

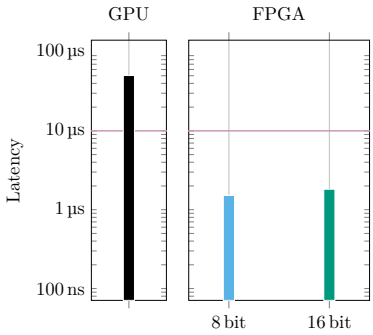
This result shows that moving Dense and Linear layers to the AIE substantially increases overall latency. The reason lies in the dataflow structure of the AIE fabric. At each PLIO interface crossing between the FPGA and AIE fabric, the preceding partition must complete its computation before the next partition can start, since the AIEs uses double-buffered interfaces. This behaviour differs from a deeply pipelined FPGA-only dataflow accelerator, where successive layers can overlap as soon as data becomes available. While AIEs supports streaming ports in principle, they do not provide the required throughput for this application scenario. Nevertheless, an overall decrease in latency is observed, since lower FPGA utilisation enables increased spatial parallelisation, as described in the following section.



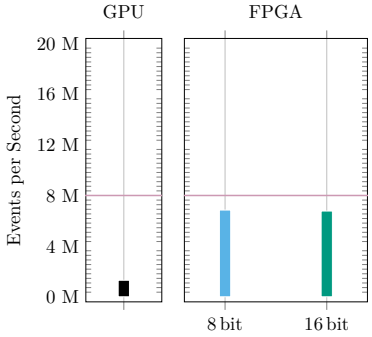
(a) Latency for $n = 32$.



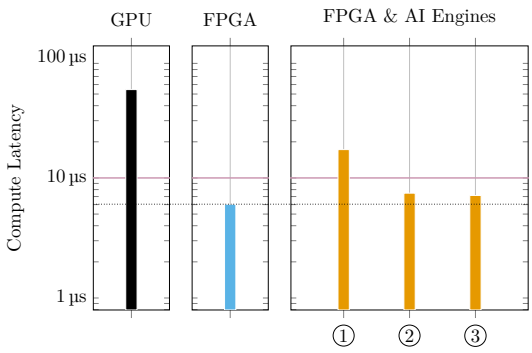
(b) Throughput for $n = 32$.



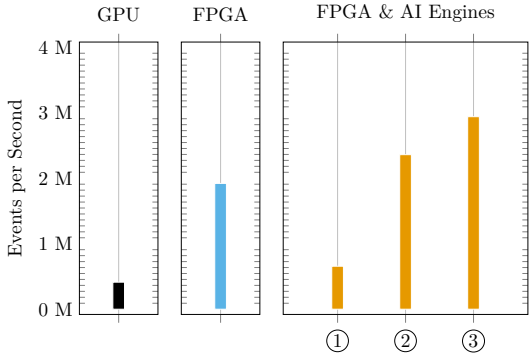
(c) Latency for $n = 64$.



(d) Throughput for $n = 64$.



(e) Latency for $n = 128$.



(f) Throughput for $n = 128$.

Figure 7.6: Performance evaluation of the proposed design for events of sizes in the range $\{32, 64, 128\}$.

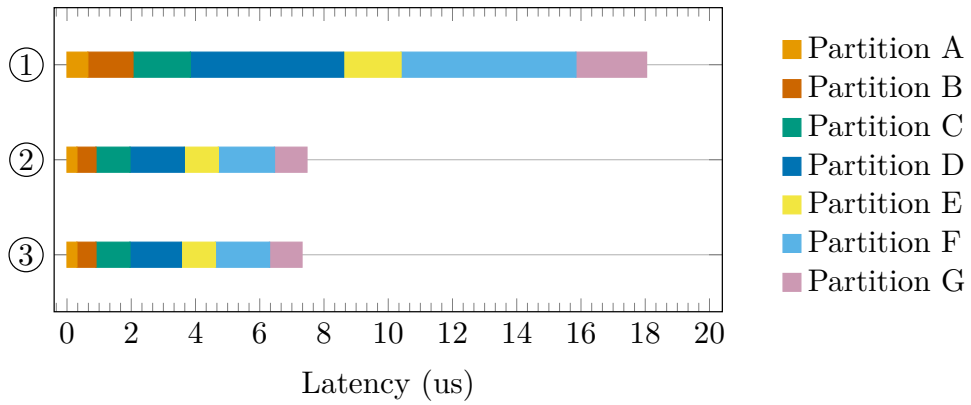


Figure 7.7: Latency breakdown for all three implementations in hardware emulation. The partition notation follows figure 7.4.

System Resource Utilization

The system resource utilisation across all eight firmware versions is reported in table 7.2. The FPGA-only implementations are evaluated at both 8 bit and 16 bit precision. The FPGA & AIE implementations use mixed precision and are shown for $n = 128$ across the three design iterations.

For the FPGA-only baselines, DSP utilisation is the limiting resource constraint. At 16 bit precision, the design saturates the available DSPs at 99.09 %, 99.24 %, and 99.14 % for $n \in \{32, 64, 128\}$, respectively. Reducing the precision to 8 bit lowers the DSP utilization to 49.75 % for $n \in \{32, 64\}$, but at $n = 128$ the DSPs are again saturated at 99.14 %. LUT utilization scales with both n and precision, ranging from 33.10 % (8 bit, $n = 32$) to 78.38 % (16 bit, $n = 64$). FF and BRAM utilization remain below 40 % in all FPGA-only configurations.

For the heterogeneous designs at $n = 128$, the partitioning shifts a substantial portion of the compute from the FPGA fabric to the AIEs. Design iteration ① reduces DSP utilization from 99.14 % (FPGA-only, 8 bit, $n = 128$) to 9.45 %, at a cost of 9.75 % of available AIE tiles. Design iterations ② and ③ share an identical resource footprint, as the kernel-level optimisations introduced in iteration ③ do not alter resource counts. Both use 18.90 % of DSPs and 29.25 % of AIE tiles, reflecting the increased spatial parallelization with $p_{\text{fpga}} = 2$ and $p_{\text{aie}} = 4$.

Generally, the growth in system resources with increasing n is expected, as the dynamic PCN layers are bound by $\mathcal{O}(n^2)$. Similarly, a large increase in DSP utilisation is expected when increasing n or the fixed-point precision of the neural network model. When not enough DSP blocks are available, the synthesis tool maps multipliers onto CLBs, drastically increasing LUT utilisation. Due to the superlinear scaling of LUTs over the fixed-point bitwidth, this is especially severe for the 16 bit versions. Overall, the FPGA-only implementations are compute-bound and constrained by the available DSP and LUT resources.

For $n = 128$, the FPGA & AIE design iteration ① achieves a 10× reduction in DSP utilization compared to the FPGA-only implementation. This enables increased spatial parallelisation for design iteration ②, which would not have been feasible without this reduction. However, even with all Dense and Linear layers deployed onto the AIE fabric, the resource bottleneck remains with the dynamic PCN layers on the FPGA partition.

Table 7.2: Resource utilisation for various design implementations on the AMD Versal VCK190 Evaluation Board. In all versions, the same neural network model from figure 7.2 is deployed.

				Programmable Logic								AI Engines					
				FF		LUT		DSP		BRAM		Tiles		Compute		Memory	
Target	Precision	n	Iter.	abs.	%	abs.	%	abs.	%	abs.	%	abs.	%	abs.	%	abs.	%
FPGA	8 bit	32	–	209 653	11.91	289 107	33.10	979	49.75	103	11.58	–	–	–	–	–	–
FPGA	16 bit	32	–	363 153	20.63	555 260	63.38	1950	99.09	133	14.89	–	–	–	–	–	–
FPGA	8 bit	64	–	296 754	16.86	391 247	44.66	979	49.75	151	16.86	–	–	–	–	–	–
FPGA	16 bit	64	–	504 146	28.64	686 633	78.38	1953	99.24	344	38.47	–	–	–	–	–	–
FPGA	8 bit	128	–	625 808	35.56	573 300	65.46	1951	99.14	170	18.98	–	–	–	–	–	–
FPGA & AIE	mixed	128	①	384 395	21.95	287 284	33.09	186	9.45	98	10.43	39	9.75	25	6.25	39	9.75
FPGA & AIE	mixed	128	②	484 210	27.51	462 972	52.85	372	18.90	292	32.63	117	29.25	84	21.00	117	29.25
FPGA & AIE	mixed	128	③	484 210	27.51	462 972	52.85	372	18.90	292	32.63	117	29.25	84	21.00	117	29.25

7.5 Discussion

In this chapter, I have presented a case study of the Belle II ECL upgrade, demonstrating the deployment of a graph-based PCN, CaloClusterNet, on a heterogeneous SoC platform. I have partitioned, optimised, and deployed the neural network on the AMD Versal VCK190 for up to 128 non-zero inputs, and validated system performance through measurements on the hardware platform. Thus, my design fulfils requirement 7.1 formulated in section 7.1. To evaluate requirements 7.2 to 7.3, latency and throughput, respectively, figure 7.8 summarises the results of the performance analysis. I conclude the following three observations:

1. Under the latency and throughput constraints of the Belle II ECL L1 Trigger, the state-of-the-art NVIDIA L40S GPUs evaluated in this work cannot meet the real-time deadline. Custom FPGA-based or heterogeneous SoC-based platforms are necessary.

2. My implementations on the AMD Versal SoC exceed the throughput of the GPU baseline by a factor of up to 7.02×. All implementations on the AMD Versal SoC fulfil the target latency requirement. This latency is non-negotiable and can not be improved through space-multiplexing.

3. For the evaluated CaloClusterNet with $n = 128$, my implementation of the accelerator fulfils the throughput requirement with a space-multiplexing factor of three.

This case study shows that the deployment methodology introduced in this thesis effectively deploys graph-based PCNs on FPGA and on AMD Versal heterogeneous SoC architectures. By introducing partitioning, operator fusion, spatial parallelisation, and kernel-level optimisations, the approach enables scaling the CaloClusterNet neural network model for large input sizes without further model compression or algorithm-level optimisations. A study shows that heterogeneous AMD Versal architectures are suitable for real-time applications when the latency deadline exceeds approximately 5 µs. As identified in this thesis, the main bottleneck to achieving higher throughput lies in the control flow of the AIE grid: double-buffered memory interfaces incur higher latency overhead than streaming interfaces in the evaluated configurations. Kernel launch overhead on the AIEs becomes significant when kernel runtimes are on the order of 100 ns, limiting the effective throughput relative to the theoretically achievable throughput. Nevertheless, combining FPGA and AIE partitions significantly improves the overall system throughput and latency.

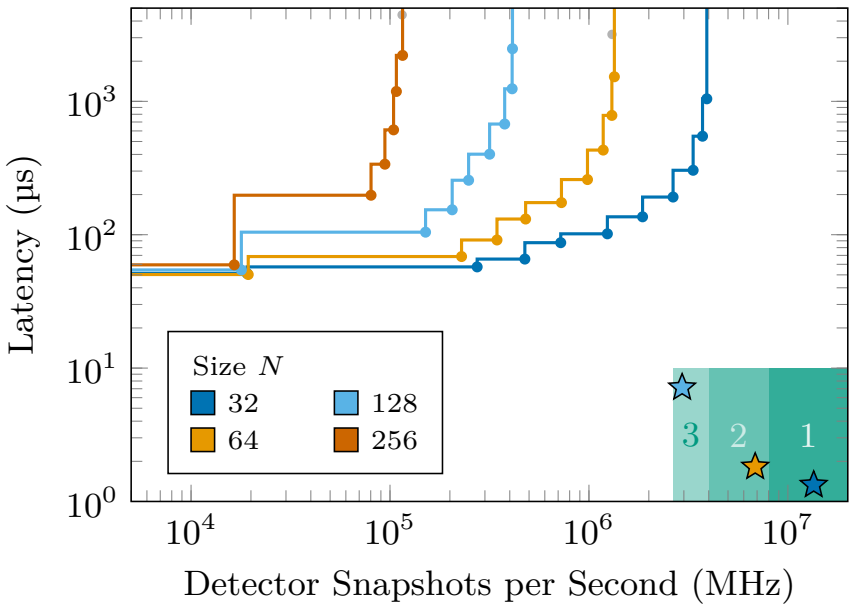


Figure 7.8: Pareto-front for the deployment of the CaloClusterNet shown in figure 7.2. Size n describes the number of non-zero inputs, typically crystals or TCs, per model inference. For the GPU baseline, measurements are shown as circles and the batch size is varied from $\{1, \ldots, 4096\}$. My implementations are shown as stars. The feasible design space of the Belle II ECL L1 Trigger is indicated by green rectangles. Numbers in the green rectangles denote the space-multiplexing factor required to achieve the target throughput of 8 million detector snapshots per second.

This page intentionally left blank

Chapter 8

Conclusion

8.1 Summary

This thesis presents the deployment of PCNs onto commercially available reprogrammable hardware through the DeePloy deployment methodology. DeePloy maps graph-based PCNs, both static and dynamic, onto heterogeneous deployment targets, ranging from FPGA-only devices to heterogeneous SoCs composed of both FPGA and CGRA partitions, with a focus on large-scale scientific experiments. This application domain imposes stringent requirements, such as hard real-time behaviour, $\mathcal{O}(1\,\mu s)$ latency, and throughput in the order of millions of events per second. In this work PCNs are decomposed into a graph of operators which are subsequently mapped onto a dataflow accelerator architecture. Transformations to the neural network formulation are proposed, based on its static or dynamic topology, to improve latency and throughput during inference. Compared with the literature this work specifically addresses the challenge of mapping graph-based PCNs onto reprogrammable hardware in a generalisable way, independent of the application-specific neural network architecture.

To evaluate DeePloy, three extensive case studies are developed, implemented, and evaluated in this thesis. First, the mapping of static PCNs onto an FPGA for the Belle II CDC L1 Trigger is evaluated for a hit-cleanup algorithm based on the Interaction Network. For this case study, a deployment architecture, the GNN-HCM, is conceptualised, implemented, and evaluated in RTL simulation. To meet the stringent system requirements, the Belle II CDC is partitioned into 20 overlapping sectors, with each sector processed by a single FPGA. The dataflow accelerator for each sector, targeting the AMD Alveo V80, achieves a latency of 0.425 µs, satisfying the hard real-time requirement of 0.5 µs, while sustaining a steady-state throughput of 32 million events per second. Compared to the previous work on the Interaction Network, this approach achieves a 9.92× higher throughput and a 4.7× lower

latency at a comparable graph size, establishing the first reference design for GNN-based hit cleanup in the Belle II CDC.

Second, the mapping of dynamic PCNs onto an FPGA for the Belle II ECL L1 Trigger is evaluated for the calorimeter clustering algorithm CaloClusterNet, which is based on GravNet. For this case study, an end-to-end deployment architecture, the GNN-ETM, is conceptualised, implemented, commissioned, and operated in the Belle II experiment. The GNN-ETM has been deployed on the UT 4, featuring an AMD UltraScale XCVU190, processing up to 32 non-zero inputs per inference, and has been operating in the Belle II L1 Trigger since the end of 2024. The end-to-end implementation achieves a latency of 1.053 µs, satisfying the hard real-time requirement of 1.067 µs, while sustaining 8 million events per second, and is ready to partake in active trigger decisions of the Belle II L1 Trigger. This constitutes the first deployment of a PCN- or GNN-based reconstruction algorithm in the hardware trigger of a particle physics experiment.

Third, the mapping of dynamic PCNs onto the AMD Versal platform is evaluated, comprising both an FPGA and a CGRA partition (AMD AIEs). For this case study, an adapted CaloClusterNet is developed to assess the scalability of the deployment approach for larger input sizes, up to 128 non-zero inputs, anticipated for the Belle II Upgrade. The dynamic PCN is optimised, partitioned, and deployed on the AMD Versal VCK190 evaluation board. Compared with the baseline GPU implementation, this deployment achieves a 7.02× throughput speedup while maintaining the hard real-time latency requirements of the Belle II L1 Trigger after the Belle II Upgrade. It thereby demonstrates, for the first time, the fully accelerated deployment of a dynamic PCN on a heterogeneous SoC for large-scale scientific experiments without any interaction with the general-purpose CPU.

To conclude, this work establishes DeePloy as a generalisable methodology for deploying static and dynamic PCNs under hard real-time constraints, spanning FPGA-only and heterogeneous SoC targets, and validated across three case studies in the Belle II experiment.

8.2 Outlook

While this thesis has demonstrated the deployment of static and dynamic PCNs across FPGA-only and heterogeneous SoC targets, several directions remain open for future work. These are grouped into extensions of the DeePloy methodology itself, further

applications within the Belle II experiment, and broader hardware and system-level directions.

Automation of the Deployment Methodology The DeePloy methodology is not yet fully automated. The mapping is performed manually, although several stages are already semi-automated through Python tooling. The DEEP collaboration[1] aims to translate the design methodology developed in this thesis into an automated flow, integrated into an MLIR-based compiler toolchain. A further direction is the integration of large-language-model-based agents into the deployment flow, for example, to assist in design-space exploration or in the generation and verification of compiler lowering passes.

Extension to Further Architectures and Compression Methods This work is limited to graph-based PCNs and their subsets, such as simple multilayer perceptrons. Novel architectures, in particular graph transformers, may be of interest for future reconstruction tasks. For transformer-based models, the attention mechanism has recently been implemented as a dataflow module on FPGA [128]. A complementary direction concerns model compression. Whereas this thesis investigates fixed-point quantisation, pruning, and power-of-two quantisation, a range of further quantisation schemes has since been developed, including high-granularity quantisation [190] and LUT-aware methods [171]. These schemes are orthogonal to the deployment approach presented here and could be incorporated, for example, by enabling the deployment of trainable layers optimised with such techniques.

Track Reconstruction in Belle II This thesis has demonstrated hit cleanup and calorimeter clustering as online reconstruction tasks. Track reconstruction, however, has not yet been demonstrated in an online trigger system and remains the most challenging of these tasks. Its realisation within the latency and throughput constraints of a hardware trigger constitutes a natural next step and is a central focus of the DEEP collaboration.

Towards Triggerless Data Acquisition Driven by the increasing compute capabilities of edge platforms and high-performance computing systems, a trend towards triggerless architectures is emerging in large-scale experiments. In such systems, the hard real-time hardware trigger is replaced by a soft real-time trigger, which may also be implemented on GPUs. Heterogeneous SoCs can play a role in this setting, providing an energy-efficient

[1] DEEP – Data Efficiency in Embedded Processors, erumdata-deep.de.

platform for quantised neural network inference as a complement to, or partial replacement for, large GPU clusters. To this end, one key enabling technology for maximising hardware efficiency in variable-sized neural network inferences on FPGAs is dynamically scheduled HLS [125].

Architectural Limitations of AMD AI Engines As shown in section 7.5, the current AMD AIE architecture exhibits limitations for batch-one inference at high throughput. Below a latency budget of approximately 5 µs, the overhead of synchronisation between AIE tiles, together with the partition crossing between the FPGA and AIE regions, becomes dominant. For small kernels with latencies close to or below 150 ns, the control-flow overhead arising from buffer locking and unlocking in the AIE tiles governs the runtime. This limitation persists in the newer AIE-ML and AIE-ML V2 architectures. This raises the question of whether custom CGRA architectures, or chiplets, should be developed for large-scale scientific experiments. Although the small market and associated costs are typically prohibitive, recent results from semiconductor start-ups indicate promising directions. Whether such low-latency CGRA concepts can be integrated as low-cost chiplets into heterogeneous SoCs remains an open question for future work.

Code Availability Statement

The software and hardware description code developed in the course of this thesis is publicly available under the repositories listed below. Each repository is associated with the chapter in which the corresponding methodology is described and with the publication in which it was first presented.

The static graph-building methodology described in section 5.3 is available at realtime-tracking/graphbuilding and was first published in Ref. [Neu24a].

The architecture template libraries described in section 4.5 are available at marcneu/pc-nhlslib and marcneu/pcnaielib, and were first published in Ref. [Neu25b] and Ref. [Neu26b], respectively.

The sparsity compression module described in section 4.6 is available at marcneu/acat-sparsity-compression and was first published in Ref. [Neu26c].

The dataflow accelerator from chapter 6 was first published in Ref. [Neu26a] and is also available as part of marcneu/pcnhlslib.

The full demonstrator implementation and evaluation for chapter 7 are available at marcneu/vck190-caloclusternet-demo and were first published in Ref. [Neu26b].

This page intentionally left blank

Bibliography

[1] Haide, Isabel: A Real-Time Graph Neural Network Trigger Algorithm for the Belle II Electromagnetic Calorimeter. 2025. doi: 10.5445/IR/1000184927. (Visited on 01/07/2026) (cit. on pp. 1, 10, 18, 25, 26, 112, 123).

[2] Reuter, Lea: "Track Finding with Graph Neural Networks in the Belle II Drift Chamber". PhD thesis. Karlsruher Institut für Technologie (KIT), 2026. doi: 10.5445/IR/1000189859. (Visited on 02/18/2026) (cit. on pp. 1, 72).

[3] Abed Abud, A.; Abi, B.; Acciarri, R.; Acero, M.A.; Adames, M.R.; Adamov, G.; Adamowski, M.; Adams, D.; Adinolfi, M. and Adriano, C.: "DUNE Phase II: Scientific Opportunities, Detector Concepts, Technological Solutions". In: *Journal of Instrumentation* 19.12 (Dec. 2024), P12005. doi: 10.1088/1748-0221/19/12/P12005. (Visited on 02/18/2026) (cit. on p. 1).

[4] Gaisser, Thomas and Halzen, Francis: "IceCube". In: *Annual Review of Nuclear and Particle Science* 64.1 (Oct. 2014), pp. 101–123. doi: 10.1146/annurev-nucl-102313-025321. (Visited on 04/12/2026) (cit. on p. 1).

[5] Abe, T. et al.: Belle II Technical Design Report. Tech. rep. arXiv, Jan. 2010. doi: 10.48550/arXiv.1011.0352 (cit. on pp. 1, 7, 13, 74).

[6] Akai, Kazunori; Furukawa, Kazuro and Koiso, Haruyo: "SuperKEKB Collider". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 907 (Nov. 2018), pp. 188–199. doi: 10.1016/j.nima.2018.08.017. (Visited on 01/07/2026) (cit. on pp. 1, 7).

[7] Collaboration, The Cms; Chatrchyan, S; Hmayakyan, G; Khachatryan, V; Sirunyan, A M; Adam, W; Bauer, T; Bergauer, T; Bergauer, H and Dragicevic, M: "The CMS Experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08004–S08004. doi: 10.1088/1748-0221/3/08/S08004. (Visited on 02/18/2026) (cit. on p. 1).

[8] Collaboration, The Atlas; Aad, G; Abat, E; Abdallah, J; Abdelalim, A A; Abdesselam, A; Abdinov, O; Abi, B A; Abolins, M and Abramowicz, H: "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08003–S08003. doi: 10.1088/1748-0221/3/08/S08003. (Visited on 02/18/2026) (cit. on p. 1).

[9] Zabi, Alexandre; Berryhill, Jeffrey Wayne; Perez, Emmanuelle and Tapper, Alexander D.: "The Phase-2 Upgrade of the CMS Level-1 Trigger". In: CERN-LHCC-2020-004, CMS-TDR-021 (2020) (cit. on p. 2).

[10] Collaboration, The Atlas: "Operation of the ATLAS Trigger System in Run 2". In: *Journal of Instrumentation* 15.10 (Oct. 2020), P10004–P10004. doi: 10.1088/1748-0221/15/10/P10004. (Visited on 02/18/2026) (cit. on p. 2).

[11] Harris, Philip et al.: Physics Community Needs, Tools, and Resources for Machine Learning. Tech. rep. FERMILAB-FN-1166-PPD-SCD, arXiv:2203.16255, 1873720. Mar. 2022, FERMILAB-FN-1166-PPD–SCD, arXiv:2203.16255, 1873720. doi: 10.2172/1873720. (Visited on 02/18/2026) (cit. on p. 2).

[12] Altarelli, Massimo; Brinkmann, R; Chergui, Majed; Decking, W; Dobson, Barry; Düsterer, Stefan; Grübel, Gerhard; Graeff, W; Graafsma, H. and Hajdu, J: The European X-Ray Free-Electron Laser. Tech. rep. DESY, Jan. 2006 (cit. on p. 3).

[13] Schroer, C. G.; Roehlsberger, R.; Weckert, E.; Wanzenberg, R.; Agapov, I.; Brinkmann, R. and Leemans, W.: PETRA IV: upgrade of PETRA III to the Ultimate 3D X-ray microscope. Conceptual Design Report. Tech. rep. DESY, 2019 (cit. on p. 3).

[14] Adams, B et al.: COMPASS++/AMBER: Proposal for Measurements at the M2 beam line of the CERN SPS Phase-1: 2022-2024. Tech. rep. CERN-SPSC-2019-022. SPSC-P-360. Geneva: CERN, May 2019. url: http://cds.cern.ch/record/2676885 (cit. on p. 3).

[15] Bediaga, I. et al.: "Framework TDR for the LHCb Upgrade: Technical Design Report". In: (Apr. 2012) (cit. on p. 3).

[16] Pilato, Christian and Soldavini, Stephanie: "Accelerator Design with High-Level Synthesis". In: *Handbook of Computer Architecture*. Ed. by Chattopadhyay, Anupam. Singapore: Springer Nature Singapore, 2025, pp. 841–873. doi: 10.1007/978-981-97-9314-3_19. (Visited on 02/09/2026) (cit. on p. 4).

[17] Reuter, L. et al.: "End-to-End Multi-track Reconstruction Using Graph Neural Networks at Belle II". In: *Computing and Software for Big Science* 9.1 (Dec. 2025), p. 6. doi: 10.1007/s41781-025-00135-6. (Visited on 02/08/2026) (cit. on pp. 4, 26, 72).

[18] Qasim, Shah Rukh; Chernyavskaya, Nadezda; Kieseler, Jan; Long, Kenneth; Viazlo, Oleksandr; Pierini, Maurizio and Nawaz, Raheel: "End-to-End Multi-Particle Reconstruction in High Occupancy Imaging Calorimeters with Graph Neural Networks". In: *The European Physical Journal C* 82.8 (Aug. 2022), p. 753. doi: 10.1140/epjc/s10052-022-10665-7. (Visited on 02/01/2026) (cit. on pp. 4, 24).

[19] Wemmer, F. et al.: "Photon Reconstruction in the Belle II Calorimeter Using Graph Neural Networks". In: *Computing and Software for Big Science* 7.1 (Dec. 2023), p. 13. doi: 10.1007/s41781-023-00105-w. (Visited on 02/01/2026) (cit. on pp. 4, 24).

[20] Gandrakota, Abhijith: Real-Time Anomaly Detection at the L1 Trigger of CMS Experiment. 2024. doi: 10.48550/ARXIV.2411.19506. (Visited on 02/21/2026) (cit. on pp. 4, 42).

[21] DeZoort, Gage; Thais, Savannah; Duarte, Javier; Razavimaleki, Vesal; Atkinson, Markus; Ojalvo, Isobel; Neubauer, Mark and Elmer, Peter: "Charged Particle Tracking via Edge-Classifying Interaction Networks". In: *Computing and Software for Big Science* 5.1 (Dec. 2021), p. 26. doi: 10.1007/s41781-021-00073-z. (Visited on 02/01/2026) (cit. on pp. 4, 22, 23).

[22] Shlomi, Jonathan; Battaglia, Peter and Vlimant, Jean-Roch: "Graph Neural Networks in Particle Physics". In: *Machine Learning: Science and Technology* 2.2 (Jan. 2021), p. 021001. doi: 10.1088/2632-2153/abbf9a. (Visited on 02/04/2026) (cit. on pp. 4, 23).

[23] Duarte, Javier and Vlimant, Jean-Roch: "Graph Neural Networks for Particle Tracking and Reconstruction". In: (2020). doi: 10.48550/ARXIV.2012.01249. (Visited on 02/21/2026) (cit. on p. 4).

[24] Kou, E. et al.: "The Belle II Physics Book". In: (2018). doi: 10.48550/ARXIV.1808.10567. (Visited on 01/07/2026) (cit. on pp. 7, 74).

[25] Martini, Alberto: "The Belle II Experiment: Status and Prospects". In: *Journal of Physics: Conference Series* 1137.1 (Jan. 2019), p. 012035. doi: 10.1088/1742-6596/1137/1/012035. (Visited on 01/07/2026) (cit. on p. 7).

[26] SuperKEKB/Belle II Complete 2024 Operations. 2024. url: https://www2.kek.jp/ipns/en/news/7015/?simply_static_page=1642969 (visited on 01/07/2026) (cit. on p. 7).

[27] Abashian, A. et al.: "The Belle Detector". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 479.1 (Feb. 2002), pp. 117–232. doi: 10.1016/S0168-9002(01)02013-7. (Visited on 01/07/2026) (cit. on pp. 7, 16).

[28] The Belle II Experiment. url: https://www.belle2.de/en/belle-2-experiment/belle-ii-detector (visited on 01/07/2026) (cit. on p. 8).

[29] Ahlburg, P. et al.: "The New and Complete Belle II DEPFET Pixel Detector: Commissioning and Previous Operational Experience". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1068 (Nov. 2024), p. 169763. doi: 10.1016/j.nima.2024.169763. url: https://linkinghub.elsevier.com/retrieve/pii/S0168900224006892 (visited on 01/07/2026) (cit. on p. 8).

[30] Ravindran, K. et al.: "Operational Experience and Performance of the Silicon Vertex Detector after the First Long Shutdown of Belle II". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1081 (Jan. 2026), p. 170834. doi: 10.1016/j.nima.2025.170834. (Visited on 01/07/2026) (cit. on p. 8).

[31] Taniguchi, N.: "Central Drift Chamber for Belle-II". In: *Journal of Instrumentation* 12.06 (June 2017), pp. C06014–C06014. doi: 10.1088/1748-0221/12/06/C06014. (Visited on 01/07/2026) (cit. on pp. 8, 13).

[32] Atmacan, H. et al.: "The Imaging Time-of-Propagation Detector at Belle II". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1080 (Nov. 2025), p. 170627. doi: 10.1016/j.nima.2025.170627. (Visited on 01/07/2026) (cit. on p. 9).

[33] Uno, Kenta: "Operation and Performance of the Belle II Aerogel RICH Detector". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1056 (Nov. 2023), p. 168635. doi: 10.1016/j.nima.2023.168635. (Visited on 01/07/2026) (cit. on p. 9).

[34] Shwartz, B. and BELLE II calorimeter group: "Electromagnetic Calorimeter of the Belle II Detector". In: *Journal of Physics: Conference Series* 928 (Nov. 2017), p. 012021. doi: 10.1088/1742-6596/928/1/012021. (Visited on 01/07/2026) (cit. on pp. 9, 16).

[35] Ketter, C. et al.: "Design and Commissioning of Readout Electronics for a K L 0 and μ Detector at the Belle II Experiment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1082 (Feb. 2026), p. 170893. doi: 10.1016/j.nima.2025.170893. (Visited on 01/07/2026) (cit. on pp. 9, 11).

[36] Aushev, T. et al.: "A Scintillator Based Endcap K and Muon Detector for the Belle II Experiment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 789 (July 2015), pp. 134–142. doi: 10.1016/j.nima.2015.03.060. (Visited on 01/07/2026) (cit. on p. 9).

[37] Yamada, Satoru; Itoh, Ryosuke; Nakamura, Katsuro; Nakao, Mikihiko; Suzuki, Soh Y.; Konno, Tomoyuki; Higuchi, Takeo; Liu, Zhen'an and Zhao, Jingzhou: "Data Acquisition System for the Belle II Experiment". In: *IEEE Transactions on Nuclear Science* 62.3 (June 2015), pp. 1175–1180. doi: 10.1109/TNS.2015.2424717. (Visited on 01/05/2026) (cit. on pp. 9, 98, 106).

[38] Sun, Dehui; Liua, Zhen'an.; Zhao, Jingzhou and Xu, Hao: "Belle2Link: A Global Data Readout and Transmission for Belle II Experiment at KEK". In: *Physics Procedia* 37 (2012), pp. 1933–1939. doi: 10.1016/j.phpro.2012.01.036. (Visited on 01/12/2026) (cit. on pp. 9, 98, 106).

[39] Higuchi, T. et al.: "Modular Pipeline Readout Electronics for the SuperBelle Drift Chamber". In: *IEEE Transactions on Nuclear Science* 52.5 (Oct. 2005), pp. 1912–1917. doi: 10.1109/TNS.2005.856913. (Visited on 01/12/2026) (cit. on p. 9).

[40] Iwasaki, Yoshihito; Cheon, ByungGu; Won, Eunil and Varner, Gary: "Level 1 Trigger System for the Belle II Experiment". In: *2010 17th IEEE-NPSS Real Time Conference*. Lisbon, Portugal: IEEE, May 2010, pp. 1–9. doi: 10.1109/RTC.2010.5750454. (Visited on 01/12/2026) (cit. on p. 10).

[41] Levit, D. et al.: "Trigger Timing Interface for the Read-Out Upgrade of the Belle II DAQ". In: *IEEE Transactions on Nuclear Science* 70.6 (June 2023), pp. 941–948. doi: 10.1109/TNS.2023.3240161. (Visited on 01/07/2026) (cit. on p. 12).

[42] Macchiarulo, L; Xin Gao; Nishimura, K and Varner, G S: "A Probability-Optimized Fast Timing Trigger for the Belle II Time of Propagation Detector". In: *IEEE Nuclear Science Symposuim & Medical Imaging Conference*. Knoxville, TN: IEEE, Oct. 2010, pp. 630–635. doi: 10.1109/NSSMIC.2010.5873835. (Visited on 01/12/2026) (cit. on p. 12).

[43] Nakao, M: "Timing Distribution for the Belle II Data Acquistion System". In: *Journal of Instrumentation* 7.01 (Jan. 2012), pp. C01028–C01028. doi: 10.1088/1748-0221/7/01/C01028. (Visited on 01/12/2026) (cit. on p. 12).

[44] Kim, C.-H. et al.: "Trigger Slow Control System of the Belle II Experiment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1014 (Oct. 2021), p. 165748. doi: 10.1016/j.nima.2021.165748. (Visited on 01/12/2026) (cit. on pp. 12, 120–122).

[45] Konno, Tomoyuki; Itoh, Ryosuke; Nakao, Mikihiko; Suzuki, Soh Y. and Yamada, Satoru: "The Slow Control and Data Quality Monitoring System for the Belle II Experiment". In:

IEEE Transactions on Nuclear Science 62.3 (June 2015), pp. 897–902. DOI: 10.1109/TNS.2015.2423091. (Visited on 01/12/2026) (cit. on p. 12).

[46] IEEE Standard for a Versatile Backplane Bus: VMEbus. ANSI/IEEE Standard. New York, NY, USA: Institute of Electrical and Electronics Engineers, 1987 (cit. on pp. 12, 98).

[47] NAKAO, M. and SUZUKI, S.Y.: "Network Shared Memory Framework for the Belle Data Acquisition Control System". In: *1999 IEEE Conference on Real-Time Computer Applications in Nuclear Particle and Plasma Physics. 11th IEEE NPSS Real Time Conference. Conference Record (Cat. No.99EX295)*. Sante Fe, NM, USA: IEEE, 1999, pp. 346–350. DOI: 10.1109/RTCON.1999.842639. (Visited on 01/12/2026) (cit. on p. 12).

[48] UCHIDA, Tomohisa; IKENO, Masahiro; IWASAKI, Yoshihito; SAITO, Masatoshi; SHIMAZAKI, Shoichi; TANAKA, Manobu; TANIGUCHI, Nanae and UNO, Shoji: "Readout Electronics for the Central Drift Chamber of the Belle II Detector". In: *2011 IEEE Nuclear Science Symposium Conference Record*. Valencia, Spain: IEEE, Oct. 2011, pp. 694–698. DOI: 10.1109/NSSMIC.2011.6154084. (Visited on 01/21/2026) (cit. on p. 13).

[49] HEINE, Greta: "Graph Neural Network based Hit Filtering for the Belle II Central Drift Chamber". PhD thesis. May 2026 (cit. on pp. 13, 72, 75, 80, 91).

[50] SHIU, J.-G.: "The Level 1 Trigger System for Belle II CDC". In: *2016 IEEE-NPSS Real Time Conference (RT)*. Padova, Italy: IEEE, June 2016, pp. 1–3. DOI: 10.1109/RTC.2016.7543137. (Visited on 01/12/2026) (cit. on p. 14).

[51] UNGER, Kai Lukas; BÄHR, Steffen; BECKER, Jürgen; IWASAKI, Yoshihito; KIM, KyungTae and LAI, Yun-Tsung: "Realization of a State Machine Based Detection for Track Segments in the Trigger System of the Belle II Experiment". In: *Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019)*. Santiago de Compostela - Spain: Sissa Medialab, Mar. 2020, p. 145. DOI: 10.22323/1.370.0145. (Visited on 01/24/2026) (cit. on pp. 14, 75).

[52] LAI, Y.-T. et al.: "Development of the Level-1 Track Trigger with Central Drift Chamber Detector in Belle II Experiment and Its Performance in SuperKEKB 2019 Phase 3 Operation". In: *Journal of Instrumentation* 15.06 (June 2020), pp. C06063–C06063. DOI: 10.1088/1748-0221/15/06/C06063. (Visited on 01/12/2026) (cit. on p. 15).

[53] LIU, Y.-X. et al.: "Development of Deep Neural Network First-Level Hardware Track Trigger for the Belle II Experiment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1084 (Apr. 2026), p. 171248. DOI: 10.1016/j.nima.2025.171248. (Visited on 01/05/2026) (cit. on p. 15).

[54] CHEON, ByungGu: "Design of a Electromagnetic Calorimeter Trigger System for the Belle II Experiment". In: *Journal of the Korean Physical Society* 57.6 (Dec. 2010), pp. 1369–1375. DOI: 10.3938/jkps.57.1369. (Visited on 01/12/2026) (cit. on pp. 17, 18).

[55] KIM, SungHyun; LEE, InSoo; UNNO, Yuuji and CHEON, ByungGu: "Status of the Electromagnetic Calorimeter Trigger System at Belle II." In: *Journal of Physics: Conference Series* 928 (Nov. 2017), p. 012022. DOI: 10.1088/1742-6596/928/1/012022. (Visited on 01/05/2026) (cit. on p. 17).

[56] Cheon, B.G et al.: "Electromagnetic Calorimeter Trigger at Belle". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 494.1-3 (Nov. 2002), pp. 548–554. doi: 10.1016/S0168-9002(02)01547-4. (Visited on 01/05/2026) (cit. on pp. 17, 106).

[57] Liang, Zhengfa; Guo, Yulan; Feng, Yiliu; Chen, Wei; Qiao, Linbo; Zhou, Li; Zhang, Jianfeng and Liu, Hengzhu: "Stereo Matching Using Multi-Level Cost Volume and Multi-Scale Feature Constancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.1 (Jan. 2021), pp. 300–315. doi: 10.1109/TPAMI.2019.2928550. (Visited on 01/28/2026) (cit. on p. 19).

[58] Guo, Yulan; Wang, Hanyun; Hu, Qingyong; Liu, Hao; Liu, Li and Bennamoun, Mohammed: "Deep Learning for 3D Point Clouds: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (Dec. 2021), pp. 4338–4364. doi: 10.1109/TPAMI.2020.3005434. (Visited on 01/28/2026) (cit. on p. 19).

[59] Bello, Saifullahi Aminu; Yu, Shangshu; Wang, Cheng; Adam, Jibril Muhmmad and Li, Jonathan: "Review: Deep Learning on 3D Point Clouds". In: *Remote Sensing* 12.11 (May 2020), p. 1729. doi: 10.3390/rs12111729. (Visited on 01/28/2026) (cit. on p. 19).

[60] Sarker, Sushmita; Sarker, Prithul; Stone, Gunner; Gorman, Ryan; Tavakkoli, Alireza; Bebis, George and Sattarvand, Javad: "A Comprehensive Overview of Deep Learning Techniques for 3D Point Cloud Classification and Semantic Segmentation". In: *Machine Vision and Applications* 35.4 (July 2024), p. 67. doi: 10.1007/s00138-024-01543-1. (Visited on 01/28/2026) (cit. on pp. 19, 20).

[61] Zhou, Wei; Liu, Kunlong; Jin, Weiwei; Wang, Qian; She, Yunfeng; Yu, Yongxiang and Ma, Caiwen: "Advancements in Deep Learning for Point Cloud Classification and Segmentation: A Comprehensive Review". In: *Computers & Graphics* 130 (Aug. 2025), p. 104238. doi: 10.1016/j.cag.2025.104238. (Visited on 01/28/2026) (cit. on p. 19).

[62] Charles, R. Qi; Su, Hao; Kaichun, Mo and Guibas, Leonidas J.: "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 77–85. doi: 10.1109/CVPR.2017.16. (Visited on 01/28/2026) (cit. on p. 19).

[63] Qi, Charles R.; Yi, Li; Su, Hao and Guibas, Leonidas J.: "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 5105–5114 (cit. on p. 19).

[64] Ma, Xu; Qin, Can; You, Haoxuan; Ran, Haoxi and Fu, Yun: Rethinking Network Design and Local Geometry in Point Cloud: A Simple Residual MLP Framework. 2022. doi: 10.48550/ARXIV.2202.07123. (Visited on 01/28/2026) (cit. on p. 19).

[65] Su, Hang; Maji, Subhransu; Kalogerakis, Evangelos and Learned-Miller, Erik: "Multi-View Convolutional Neural Networks for 3D Shape Recognition". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 945–953. doi: 10.1109/ICCV.2015.114. (Visited on 01/28/2026) (cit. on p. 20).

[66] Feng, Yifan; Zhang, Zizhao; Zhao, Xibin; Ji, Rongrong and Gao, Yue: "GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 264–272. doi: 10.1109/CVPR.2018.00035. (Visited on 01/28/2026) (cit. on p. 20).

[67] Woo, Sungmin; Lee, Dogyoon; Hwang, Sangwon; Kim, Woo Jin and Lee, Sangyoun: "MK-Conv: Multidimensional Feature Representation for Point Cloud Analysis". In: *Pattern Recognition* 143 (Nov. 2023), p. 109800. doi: 10.1016/j.patcog.2023.109800. (Visited on 01/28/2026) (cit. on p. 20).

[68] Zhou, Wei; Jin, Weiwei; Wang, Dekui; Hao, Xingxing; Yu, Yongxiang and Ma, Caiwen: "Exploring Multi-Scale and Cross-Type Features in 3D Point Cloud Learning with CCMNET". In: *Expert Systems with Applications* 274 (May 2025), p. 126960. doi: 10.1016/j.eswa.2025.126960. (Visited on 01/28/2026) (cit. on p. 20).

[69] Wang, Yue; Sun, Yongbin; Liu, Ziwei; Sarma, Sanjay E.; Bronstein, Michael M. and Solomon, Justin M.: "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Transactions on Graphics* 38.5 (Oct. 2019), pp. 1–12. doi: 10.1145/3326362. (Visited on 02/01/2026) (cit. on pp. 20, 22, 37).

[70] Qu, Huilin and Gouskos, Loukas: "Jet Tagging via Particle Clouds". In: *Physical Review D* 101.5 (Mar. 2020), p. 056019. doi: 10.1103/PhysRevD.101.056019. (Visited on 02/01/2026) (cit. on pp. 20, 21).

[71] Li, Dilong; Lu, Chenghui; Chen, Ziyi; Guan, Jianlong; Zhao, Jing and Du, Jixiang: "Graph Neural Networks in Point Clouds: A Survey". In: *Remote Sensing* 16.14 (July 2024), p. 2518. doi: 10.3390/rs16142518. (Visited on 01/28/2026) (cit. on p. 21).

[72] Gilmer, Justin; Schoenholz, Samuel S.; Riley, Patrick F.; Vinyals, Oriol and Dahl, George E.: "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1263–1272 (cit. on p. 21).

[73] Fey, Matthias and Lenssen, Jan Eric: Fast Graph Representation Learning with PyTorch Geometric. 2019. doi: 10.48550/ARXIV.1903.02428. (Visited on 02/04/2026) (cit. on pp. 21, 88, 157).

[74] Qasim, Shah Rukh; Kieseler, Jan; Iiyama, Yutaro and Pierini, Maurizio: "Learning Representations of Irregular Particle-Detector Geometry with Distance-Weighted Graph Networks". In: *The European Physical Journal C* 79.7 (July 2019), p. 608. doi: 10.1140/epjc/s10052-019-7113-9. (Visited on 02/04/2026) (cit. on pp. 22, 24).

[75] Wei, Mingqiang et al.: "AGConv: Adaptive Graph Convolution on 3D Point Clouds". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.8 (Aug. 2023), pp. 9374–9392. doi: 10.1109/TPAMI.2023.3238516. (Visited on 02/01/2026) (cit. on p. 22).

[76] Battaglia, Peter; Pascanu, Razvan; Lai, Matthew; Rezende, Danilo Jimenez and kavukcuoglu, Koray: "Interaction Networks for Learning about Objects, Relations and Physics". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 4509–4517 (cit. on p. 23).

[77] Battaglia, Peter W. et al.: Relational Inductive Biases, Deep Learning, and Graph Networks. 2018. doi: 10.48550/ARXIV.1806.01261. (Visited on 02/04/2026) (cit. on p. 23).

[78] Iiyama, Yutaro et al.: "Distance-Weighted Graph Neural Networks on FPGAs for Real-Time Particle Reconstruction in High Energy Physics". In: *Frontiers in Big Data* 3 (Jan. 2021), p. 598927. doi: 10.3389/fdata.2020.598927. (Visited on 02/01/2026) (cit. on pp. 24, 38, 43).

[79] Vaidya, Pravin M.: "AnO(n Logn) Algorithm for the All-Nearest-Neighbors Problem". In: *Discrete & Computational Geometry* 4.2 (Mar. 1989), pp. 101–115. doi: 10.1007/BF02187718. (Visited on 02/04/2026) (cit. on pp. 25, 50, 91).

[80] Kieseler, Jan: "Object Condensation: One-Stage Grid-Free Multi-Object Reconstruction in Physics Detectors, Graph, and Image Data". In: *The European Physical Journal C* 80.9 (Sept. 2020), p. 886. doi: 10.1140/epjc/s10052-020-08461-2. (Visited on 02/01/2026) (cit. on pp. 25, 94).

[81] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg and Xu, Xiaowei: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231 (cit. on p. 25).

[82] Amano, Hideharu, ed.: Principles and Structures of FPGAs. Singapore: Springer Singapore, 2018. doi: 10.1007/978-981-13-0824-6. (Visited on 02/08/2026) (cit. on p. 27).

[83] Hauck, Scott and DeHon, Andre: Reconfigurable Computing: The Theory and Practice of FPGA-based Computation. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007 (cit. on p. 27).

[84] Pfau, Johannes: RFET Reconfigurable Devices: Power Aware FPGA Architectures and Toolflow. 2024. doi: 10.5445/IR/1000174452. (Visited on 02/11/2026) (cit. on p. 28).

[85] Boutros, Andrew and Betz, Vaughn: "FPGA Architecture: Principles and Progression". In: *IEEE Circuits and Systems Magazine* 21.2 (2021), pp. 4–29. doi: 10.1109/MCAS.2021.3071607. (Visited on 02/08/2026) (cit. on pp. 27, 32).

[86] Chen, Shixin; Zhang, Hengyuan; Ling, Zichao; Zhai, Jianwang and Yu, Bei: "The Survey of 2.5D Integrated Architecture: An EDA Perspective". In: *Proceedings of the 30th Asia and South Pacific Design Automation Conference*. Tokyo Japan: ACM, Jan. 2025, pp. 285–293. doi: 10.1145/3658617.3703134. (Visited on 02/11/2026) (cit. on pp. 29, 32).

[87] Madden, Liam; Wu, Ephrem; Kim, Namhoon; Banijamali, Bahareh; Abugharbieh, Khaldoon; Ramalingam, Suresh and Wu, Xin: "Advancing High Performance Heterogeneous Integration through Die Stacking". In: *2012 Proceedings of the ESSCIRC (ESSCIRC)*. Bordeaux, France: IEEE, Sept. 2012, pp. 18–24. doi: 10.1109/ESSCIRC.2012.6341246. (Visited on 02/11/2026) (cit. on p. 29).

[88] Wu, Ephrem; Abugharbieh, Khaldoon; Banijamali, Bahareh; Ramalingam, Suresh; Wu, Paul and Wyland, Chris: "Interconnect and Package Design of a Heterogeneous Stacked-Silicon FPGA". In: *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*. San Jose, CA, USA: IEEE, Sept. 2013, pp. 1–8. doi: 10.1109/CICC.2013.6658402. (Visited on 02/11/2026) (cit. on p. 29).

[89] AMD: Versal Adaptive SoC Design Guide. Manual. 2026 (cit. on pp. 29, 32).

[90] AMD: Ultrascale Architecture Configuration User Guide. Manual. 2026 (cit. on p. 29).

[91] Gaide, Brian; Gaitonde, Dinesh; Ravishankar, Chirag and Bauer, Trevor: "Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture". In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Seaside CA USA: ACM, Feb. 2019, pp. 84–93. doi: 10.1145/3289602.3293906. (Visited on 02/11/2026) (cit. on pp. 29, 30, 32, 33).

[92] Kuon, Ian and Rose, Jonathan: "Measuring the Gap Between FPGAs and ASICs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.2 (Feb. 2007), pp. 203–215. doi: 10.1109/TCAD.2006.884574. (Visited on 02/11/2026) (cit. on pp. 30, 145).

[93] Liu, Leibo; Zhu, Jianfeng; Li, Zhaoshi; Lu, Yanan; Deng, Yangdong; Han, Jie; Yin, Shouyi and Wei, Shaojun: "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications". In: *ACM Computing Surveys* 52.6 (Nov. 2020), pp. 1–39. doi: 10.1145/3357375. (Visited on 02/11/2026) (cit. on p. 30).

[94] Chin, S. Alexander; Sakamoto, Noriaki; Rui, Allan; Zhao, Jim; Kim, Jin Hee; Hara-Azumi, Yuko and Anderson, Jason: "CGRA-ME: A Unified Framework for CGRA Modelling and Exploration". In: *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. Seattle, WA, USA: IEEE, July 2017, pp. 184–189. doi: 10.1109/ASAP.2017.7995277. (Visited on 02/11/2026) (cit. on pp. 30, 31).

[95] Podobas, Artur; Sano, Kentaro and Matsuoka, Satoshi: "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective". In: *IEEE Access* 8 (2020), pp. 146719–146743. doi: 10.1109/ACCESS.2020.3012084. (Visited on 02/11/2026) (cit. on p. 30).

[96] Hartenstein, R.W.; Hirschbiel, A.G.; Riedmuller, M.; Schmidt, K. and Weber, M.: "A Novel ASIC Design Approach Based on a New Machine Paradigm". In: *IEEE Journal of Solid-State Circuits* 26.7 (July 1991), pp. 975–989. doi: 10.1109/4.92017. (Visited on 02/11/2026) (cit. on p. 30).

[97] Wei, Shaojun; Lin, Xinhan; Tu, Fengbin; Wang, Yang; Liu, Leibo and Yin, Shouyi: "Reconfigurability, Why It Matters in AI Tasks Processing: A Survey of Reconfigurable AI Chips". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 70.3 (Mar. 2023), pp. 1228–1241. doi: 10.1109/TCSI.2022.3228860. (Visited on 02/11/2026) (cit. on p. 30).

[98] Hennessy, John L. and Patterson, David A.: Computer Architecture: A Quantitative Approach. Sixth edition. Cambridge, MA: Morgan Kaufmann Publishers, 2019 (cit. on p. 31).

[99] Sze, Vivienne; Chen, Yu-Hsin; Yang, Tien-Ju and Emer, Joel S.: Efficient Processing of Deep Neural Networks. Synthesis Lectures on Computer Architecture. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-031-01766-7. (Visited on 02/14/2026) (cit. on p. 32).

[100] Baruah, S.: "Task Partitioning upon Heterogeneous Multiprocessor Platforms". In: *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications*

Symposium, 2004. Toronto, Canada: IEEE, 2004, pp. 536–543. DOI: 10.1109/RTTAS.2004.1317301. (Visited on 02/14/2026) (cit. on p. 32).

[101] ARM LTD.: ARM Cortex-A72 Mpcore Processor Technical Reference Manual. Manual. 2016 (cit. on p. 32).

[102] ARM LTD.: ARM Cortex-R5 and Cortex-R5F Technical Reference Manual. Manual. 2011 (cit. on p. 32).

[103] KININGHAM, Kevin; LEVIS, Philip and RÉ, Christopher: "GRIP: A Graph Neural Network Accelerator Architecture". In: *IEEE Transactions on Computers* 72.4 (Apr. 2023), pp. 914–925. DOI: 10.1109/TC.2022.3197083. (Visited on 02/25/2026) (cit. on pp. 36, 43).

[104] YANG, Tao; LI, Dongyue; HAN, Yibo; ZHAO, Yilong; LIU, Fangxin; LIANG, Xiaoyao; HE, Zhezhi and JIANG, Li: "PIMGCN: A ReRAM-Based PIM Design for Graph Convolutional Network Acceleration". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 583–588. DOI: 10.1109/DAC18074.2021.9586231. (Visited on 02/25/2026) (cit. on pp. 36, 43).

[105] ZHANG, Bingyi; KANNAN, Rajgopal and PRASANNA, Viktor: "A Framework for Optimizing GCN Inference on FPGA". In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 145–145. DOI: 10.1145/3431920.3439456. (Visited on 02/25/2026) (cit. on pp. 36, 43).

[106] LIANG, Shengwen; WANG, Ying; LIU, Cheng; HE, Lei; LI, Huawei; XU, Dawen and LI, Xiaowei: "EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks". In: *IEEE Transactions on Computers* 70.9 (Sept. 2021), pp. 1511–1525. DOI: 10.1109/TC.2020.3014632. (Visited on 02/25/2026) (cit. on p. 36).

[107] SOHRABIZADEH, Atefeh; CHI, Yuze and CONG, Jason: "StreamGCN: Accelerating Graph Convolutional Networks with Streaming Processing". In: *2022 IEEE Custom Integrated Circuits Conference (CICC)*. Newport Beach, CA, USA: IEEE, Apr. 2022, pp. 1–8. DOI: 10.1109/CICC53496.2022.9772832. (Visited on 02/25/2026) (cit. on pp. 36, 43).

[108] ZHANG, Chengming; GENG, Tong; GUO, Anqi; TIAN, Jiannan; HERBORDT, Martin; LI, Ang and TAO, Dingwen: "H-GCN: A Graph Convolutional Network Accelerator on Versal ACAP Architecture". In: *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. Belfast, United Kingdom: IEEE, Aug. 2022, pp. 200–208. DOI: 10.1109/FPL57034.2022.00040. (Visited on 02/25/2026) (cit. on pp. 36, 43).

[109] HEINTZ, Aneesh et al.: Accelerated Charged Particle Tracking with Graph Neural Networks on FPGAs. Tech. rep. arXiv, Jan. 2020. DOI: 10.48550/arXiv.2012.01563 (cit. on p. 36).

[110] ELABD, Abdelrahman et al.: "Graph Neural Networks for Charged Particle Tracking on FPGAs". In: *Frontiers in Big Data* 5 (Mar. 2022), p. 828666. DOI: 10.3389/fdata.2022.828666. (Visited on 02/04/2026) (cit. on p. 36).

[111] HUANG, Shi-Yu; YANG, Yun-Chen; SU, Yu-Ru; LAI, Bo-Cheng; DUARTE, Javier; HAUCK, Scott; HSU, Shih-Chieh; HU, Jin-Xuan and NEUBAUER, Mark S.: "Low Latency Edge Classification GNN for Particle Trajectory Tracking on FPGAs". In: *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. Gothenburg, Sweden: IEEE, Sept.

2023, pp. 294–298. DOI: 10.1109/FPL60245.2023.00050. (Visited on 02/25/2026) (cit. on pp. 36, 43, 92).

[112] QUE, Zhiqiang; FAN, Hongxiang; LOO, Marcus; LI, He; BLOTT, Michaela; PIERINI, Maurizio; TAPPER, Alexander and LUK, Wayne: "LL-GNN: Low Latency Graph Neural Networks on FPGAs for High Energy Physics". In: *ACM Transactions on Embedded Computing Systems* 23.2 (Mar. 2024), pp. 1–28. DOI: 10.1145/3640464. (Visited on 02/25/2026) (cit. on p. 37).

[113] ODAGIU, Patrick et al.: "Ultrafast Jet Classification at the HL-LHC". In: *Machine Learning: Science and Technology* 5.3 (Sept. 2024), p. 035017. DOI: 10.1088/2632-2153/ad5f10. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[114] QUE, Zhiqiang et al.: JEDI-linear: Fast and Efficient Graph Neural Networks for Jet Tagging on FPGAs. 2025. DOI: 10.48550/ARXIV.2508.15468. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[115] LIN, Yujun; ZHANG, Zhekai; TANG, Haotian; WANG, Hanrui and HAN, Song: "PointAcc: Efficient Point Cloud Accelerator". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Virtual Event Greece: ACM, Oct. 2021, pp. 449–461. DOI: 10.1145/3466752.3480084. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[116] ZHANG, Jie-Fang and ZHANG, Zhengya: "Point-X: A Spatial-Locality-Aware Architecture for Energy-Efficient Graph-Based Point-Cloud Deep Learning". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Virtual Event Greece: ACM, Oct. 2021, pp. 1078–1090. DOI: 10.1145/3466752.3480081. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[117] MONDAL, Sudipta and SAPATNEKAR, Sachin S.: "Hardware Acceleration of Inference on Dynamic GNNs". In: *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. Newport Beach CA USA: ACM, Aug. 2024, pp. 1–6. DOI: 10.1145/3665314.3670832. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[118] LIANG, Shengwen; LIU, Cheng; WANG, Ying; LI, Huawei and LI, Xiaowei: "DeepBurning-GL: An Automated Framework for Generating Graph Neural Network Accelerators". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. Virtual Event USA: ACM, Nov. 2020, pp. 1–9. DOI: 10.1145/3400302.3415645. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[119] JAMALI GOLZAR, Saleh; KARIMIAN, Ghader; SHOARAN, Maryam and FATTAHI SANI, Mohammad: "DGCNN on FPGA: Acceleration of the Point Cloud Classifier Using FPGAs". In: *Circuits, Systems, and Signal Processing* 42.2 (Feb. 2023), pp. 748–779. DOI: 10.1007/s00034-022-02179-0. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[120] CHEN, Hanqiu and HAO, Cong: DGNN-Booster: A Generic FPGA Accelerator Framework For Dynamic Graph Neural Network Inference. 2023. DOI: 10.48550/ARXIV.2304.06831. (Visited on 02/25/2026) (cit. on pp. 37, 43).

[121] GAO, Yiming; JIANG, Chao; PIARD, Wesley; CHEN, Xiangru; PATEL, Bhavesh and LAM, Herman: "HgPCN: A Heterogeneous Architecture for E2E Embedded Point Cloud Inference". In: *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Austin,

TX, USA: IEEE, Nov. 2024, pp. 1588–1600. DOI: 10.1109/MICRO61859.2024.00116. (Visited on 02/25/2026) (cit. on pp. 37, 42, 43).

[122] AMD/Xilinx: Vitis High-Level Synthesis User Guide (UG1399). Manual. 2026 (cit. on p. 38).

[123] Canis, Andrew; Choi, Jongsok; Aldham, Mark; Zhang, Victor; Kammoona, Ahmed; Anderson, Jason H.; Brown, Stephen and Czajkowski, Tomasz: "LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems". In: *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Monterey CA USA: ACM, Feb. 2011, pp. 33–36. DOI: 10.1145/1950413.1950423. (Visited on 03/04/2026) (cit. on p. 38).

[124] Bollaert, Thomas: "Catapult Synthesis: A Practical Introduction to Interactive C Synthesis". In: *High-Level Synthesis*. Ed. by Coussy, Philippe and Morawiec, Adam. Dordrecht: Springer Netherlands, 2008, pp. 29–52. DOI: 10.1007/978-1-4020-8588-8_3. (Visited on 03/04/2026) (cit. on p. 38).

[125] Josipović, Lana; Ghosal, Radhika and Ienne, Paolo: "Dynamically Scheduled High-level Synthesis". In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Monterey CALIFORNIA USA: ACM, Feb. 2018, pp. 127–136. DOI: 10.1145/3174243.3174264. (Visited on 10/26/2023) (cit. on pp. 38, 170).

[126] Umuroglu, Yaman; Fraser, Nicholas J.; Gambardella, Giulio; Blott, Michaela; Leong, Philip; Jahre, Magnus and Vissers, Kees: "FINN". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Ed. by Greene, Jonathan and Anderson, Jason H. New York, NY, USA: ACM, Jan. 2017, pp. 65–74. DOI: 10.1145/3020078.3021744 (cit. on pp. 39, 43, 47, 48, 62).

[127] Blott, Michaela; Preusser, Thomas B.; Fraser, Nicholas J.; Gambardella, Giulio; O'brien, Kenneth; Umuroglu, Yaman; Leeser, Miriam and Vissers, Kees: "FINN- R". In: *ACM Transactions on Reconfigurable Technology and Systems* 11.3 (Jan. 2018), pp. 1–23. DOI: 10.1145/3242897 (cit. on pp. 39, 47, 48, 62).

[128] Berganski, Christoph; Jentzsch, Felix; Platzner, Marco; Kuhmichel, Max and Giefers, Heiner: "FINN-T: Compiling Custom Dataflow Accelerators for Quantized Transformers". In: *2024 International Conference on Field Programmable Technology (ICFPT)*. Sydney, Australia: IEEE, Dec. 2024, pp. 01–10. DOI: 10.1109/ICFPT64416.2024.11113391. (Visited on 02/25/2026) (cit. on pp. 39, 169).

[129] Schulte, Jan-Frederik et al.: Hls4ml: A Flexible, Open-Source Platform for Deep Learning Acceleration on Reconfigurable Hardware. 2025. DOI: 10.48550/ARXIV.2512.01463. (Visited on 02/18/2026) (cit. on p. 39).

[130] Team, FastML: Hls4ml. Zenodo. Jan. 2023. DOI: 10.5281/zenodo.1201549 (cit. on pp. 39, 43, 47, 48, 62, 114).

[131] Abi-Karam, Stefan and Hao, Cong: "GNNBuilder: An Automated Framework for Generic Graph Neural Network Accelerator Generation, Simulation, and Optimization". In: (2023). DOI: 10.48550/ARXIV.2303.16459. (Visited on 09/18/2023) (cit. on pp. 39, 42, 43, 59).

[132] Sarkar, Rishov; Abi-Karam, Stefan; He, Yuqi; Sathidevi, Lakshmi and Hao, Cong: "FlowGNN: A Dataflow Architecture for Real-Time Workload-Agnostic Graph Neural Network Inference". In: *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Montreal, QC, Canada: IEEE, Feb. 2023, pp. 1099–1112. doi: 10.1109/HPCA56546.2023.10071015. (Visited on 09/20/2023) (cit. on pp. 39, 42, 43).

[133] Taka, Endri; Arora, Aman; Wu, Kai-Chiang and Marculescu, Diana: "MaxEVA: Maximizing the Efficiency of Matrix Multiplication on Versal AI Engine". In: *2023 International Conference on Field Programmable Technology (ICFPT)*. Dec. 2023, pp. 96–105. doi: 10.1109/ICFPT59805.2023.00016. (Visited on 11/16/2025) (cit. on pp. 39, 43).

[134] Taka, Endri; Gourounas, Dimitrios; Gerstlauer, Andreas; Marculescu, Diana and Arora, Aman: "Efficient Approaches for GEMM Acceleration on Leading AI-Optimized FPGAs". In: *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. May 2024, pp. 54–65. doi: 10.1109/FCCM60383.2024.00015. (Visited on 11/16/2025) (cit. on p. 39).

[135] Deng, Xiaodong; Wang, Shijie; Gao, Tianyi; Liu, Jing; Liu, Longjun and Zheng, Nanning: "AMA: An Analytical Approach to Maximizing the Efficiency of Deep Learning on Versal AI Engine". In: *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. Sept. 2024, pp. 227–235. doi: 10.1109/FPL64840.2024.00039. (Visited on 12/10/2025) (cit. on pp. 40, 43).

[136] Mhatre, Kaustubh; Taka, Endri and Arora, Aman: GAMA: High-Performance GEMM Acceleration on AMD Versal ML-Optimized AI Engines. Aug. 2025. doi: 10.48550/arXiv.2504.09688. arXiv: 2504.09688 [cs]. (Visited on 11/16/2025) (cit. on pp. 40, 43).

[137] Danopoulos, Dimitrios; Lupi, Enrico; Sun, Chang; Dittmeier, Sebastian; Kagan, Michael; Loncar, Vladimir and Pierini, Maurizio: AIE4ML: An End-to-End Framework for Compiling Neural Networks for the Next Generation of AMD AI Engines. 2025. doi: 10.48550/ARXIV.2512.15946. (Visited on 01/19/2026) (cit. on pp. 40, 43).

[138] Abarajithan, G; Ma, Zhenghua; Munasinghe, Ravidu; Restuccia, Francesco and Kastner, Ryan: CGRA4ML: A Hardware/Software Framework to Implement Neural Networks for Scientific Edge Computing. 2024. doi: 10.48550/ARXIV.2408.15561. (Visited on 02/25/2026) (cit. on pp. 40, 43).

[139] Ye, Hanchen; Jun, HyeGang; Jeong, Hyunmin; Neuendorffer, Stephen and Chen, Deming: "ScaleHLS". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. Ed. by Oshana, Rob. New York, NY, USA: ACM, Jan. 2022, pp. 1355–1358. doi: 10.1145/3489517.3530631 (cit. on pp. 40, 43).

[140] Ye, Hanchen; Jun, Hyegang and Chen, Deming: "HIDA: A Hierarchical Dataflow Compiler for High-Level Synthesis". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. La Jolla CA USA: ACM, Apr. 2024, pp. 215–230. doi: 10.1145/3617232.3624850. (Visited on 02/25/2026) (cit. on p. 40).

[141] Levental, Maksim; Khan, Arham; Chard, Ryan; Yoshii, Kazutomo; Chard, Kyle and Foster, Ian: "BraggHLS: High-Level Synthesis for Low-Latency Deep Neural Networks for

Experimental Science". In: *14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 24)*. Porto Portugal: ACM, June 2024, pp. 10–17. DOI: 10.1145/3665283.3665284. (Visited on 12/22/2025) (cit. on pp. 40, 43).

[142] LUO, Yixuan; TAN, Cheng; AGOSTINI, Nicolas Bohm; LI, Ang; TUMEO, Antonino; DAVE, Nirav and GENG, Tong: "ML-CGRA: An Integrated Compilation Framework to Enable Efficient Machine Learning Acceleration on CGRAs". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, July 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247873. (Visited on 02/25/2026) (cit. on pp. 41, 43).

[143] LEVENTAL, Maksim: "An End-to-End Programming Model for AI Engine Architectures". PhD thesis. The University of Chicago, 2024. DOI: 10.6082/UCHICAGO.12382. (Visited on 12/22/2025) (cit. on p. 41).

[144] ZHUANG, Jinming; XIANG, Shaojie; CHEN, Hongzheng; ZHANG, Niansong; YANG, Zhuoping; MAO, Tony; ZHANG, Zhiru and ZHOU, Peipei: "ARIES: An Agile MLIR-Based Compilation Flow for Reconfigurable Devices with AI Engines". In: *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. FPGA '25. New York, NY, USA: Association for Computing Machinery, Feb. 2025, pp. 92–102. DOI: 10.1145/3706628.3708870. (Visited on 12/10/2025) (cit. on pp. 41, 43).

[145] WANG, Erwei et al.: From Loop Nests to Silicon: Mapping AI Workloads onto AMD NPUs with MLIR-AIR. Oct. 2025. DOI: 10.48550/arXiv.2510.14871. arXiv: 2510.14871 [cs]. (Visited on 12/15/2025) (cit. on pp. 41, 43).

[146] ARM LIMITED: AMBA AXI-Stream Protocol Specification. IHI 0051B. 2021. URL: https://developer.arm.com/documentation/ihi0051/latest/ (cit. on pp. 46, 98).

[147] ANSEL, Jason et al.: "PyTorch 2: Faster Machine Learning through Dynamic Python Bytecode Transformation and Graph Compilation". In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366 (cit. on pp. 47, 88, 157).

[148] DEVELOPERS, ONNX Runtime: ONNX Runtime. https://onnxruntime.ai/. 2026. (Visited on 06/02/2026) (cit. on p. 47).

[149] CALLAHAN, Paul B. and KOSARAJU, S. Rao: "A Decomposition of Multidimensional Point Sets with Applications to k -Nearest-Neighbors and n -Body Potential Fields". In: *Journal of the ACM* 42.1 (Jan. 1995), pp. 67–90. DOI: 10.1145/200836.200853. (Visited on 05/06/2026) (cit. on p. 50).

[150] CONNOR, M. and KUMAR, P.: Parallel Construction of K-Nearest Neighbor Graphs for Point Clouds. 2008. DOI: 10.2312/VG/VG-PBG08/025-031. (Visited on 05/06/2026) (cit. on pp. 50, 91).

[151] HARWOOD, Ben and DRUMMOND, Tom: "FANNG: Fast Approximate Nearest Neighbour Graphs". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jan. 2016, pp. 5713–5722. DOI: 10.1109/CVPR.2016.616 (cit. on p. 50).

[152] HAJEBI, Kiana; ABBASI-YADKORI, Yasin; SHAHBAZI, Hossein and ZHANG, Hong: "Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI-11:*

Barcelona, Catalonia, Spain, 16 - 22 July 2011. Vol. 2. Menlo Park, Calif.: AAAI Press, Jan. 2011 (cit. on p. 50).

[153] Prokhorenkova, Liudmila and Shekhovtsov, Aleksandr: "Graph-Based Nearest Neighbor Search: From Practice to Theory". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by III, Hal Daumé and Singh, Aarti. Proceedings of Machine Learning Research. PMLR, Jan. 2020, p. 78037813 (cit. on pp. 51, 77).

[154] Liu, Kenneth; Lu, Alec; Samtani, Kartik; Fang, Zhenman and Guo, Licheng: "CHIP-KNNv2: A C Onfigurable and Hi Gh- P Erformance K - N Earest N Eighbors Accelerator on HBM-based FPGAs". In: *ACM Transactions on Reconfigurable Technology and Systems* (Sept. 2023), p. 3616873. DOI: 10.1145/3616873. (Visited on 10/13/2023) (cit. on p. 59).

[155] Bachrach, Jonathan et al.: "Chisel: Constructing Hardware in a Scala Embedded Language". In: *Proceedings of the 49th Annual Design Automation Conference*. San Francisco California: ACM, June 2012, pp. 1216–1225. DOI: 10.1145/2228360.2228584. (Visited on 05/13/2025) (cit. on pp. 61, 88, 101, 105, 106, 110).

[156] AMD: Vitis Unified Software Platform. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html. Version 2024.2, accessed 2026-02-10. 2026 (cit. on pp. 61, 88, 110, 124).

[157] AMD/Xilinx: VAI Engine Kernel and Graph Programming Guide (UG1079). Manual. 2026 (cit. on pp. 62, 63).

[158] Papaphilippou, Philippos; Luk, Wayne and Gregg, David: "Efficient Adaptable Streaming Aggregation Engine". In: *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Fayetteville, AR, USA: IEEE, May 2025, pp. 297–297. DOI: 10.1109/FCCM62733.2025.00016. (Visited on 12/22/2025) (cit. on p. 68).

[159] Oge, Yasin; Yoshimi, Masato; Miyoshi, Takefumi; Kawashima, Hideyuki; Irie, Hidetsugu and Yoshinaga, Tsutomu: "An Efficient and Scalable Implementation of Sliding-Window Aggregate Operator on FPGA". In: *2013 First International Symposium on Computing and Networking*. Matsuyama, Japan: IEEE, Dec. 2013, pp. 112–121. DOI: 10.1109/CANDAR.2013.23. (Visited on 01/05/2026) (cit. on p. 68).

[160] Yoshii, Kazutomo; Ueno, Tomohiro; Sano, Kentaro; Miceli, Antonino and Cappello, Franck: "Streaming Hardware Compressor Generator Framework". In: *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. Denver CO USA: ACM, Nov. 2023, pp. 289–297. DOI: 10.1145/3624062.3625126. (Visited on 01/05/2026) (cit. on p. 68).

[161] Agostinelli, S. et al.: "Geant4—a Simulation Toolkit". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (July 2003), pp. 250–303. DOI: 10.1016/S0168-9002(03)01368-8. (Visited on 05/06/2026) (cit. on p. 74).

[162] Kuhr, T.; Pulvermacher, C.; Ritter, M.; Hauth, T. and Braun, N.: "The Belle II Core Software: Belle II Framework Software Group". In: *Computing and Software for Big Science*

3.1 (Dec. 2019), p. 1. DOI: 10.1007/s41781-018-0017-9. (Visited on 05/06/2026) (cit. on p. 74).

[163] LIPTAK, Z. et al.: "Measurements of Beam Backgrounds in SuperKEKB Phase 2". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1040 (Oct. 2022), p. 167168. DOI: 10.1016/j.nima.2022.167168. (Visited on 05/06/2026) (cit. on p. 74).

[164] NATOCHII, A. et al.: "Beam Background Expectations for Belle II at SuperKEKB". In: *Snowmass 2021*. Mar. 2022 (cit. on p. 74).

[165] POHL, Sara: "Track Reconstruction at the First Level Trigger of the Belle II Experiment". PhD thesis. Jan. 2018. DOI: 10.5282/edoc.22085 (cit. on p. 75).

[166] IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. Tech. rep. Piscataway, NJ, USA: IEEE, 2017. DOI: 10.1109/IEEESTD.2018.8299595 (cit. on p. 85).

[167] AMD: Vivado Design Suite. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html. Version 2024.2, accessed 2026-02-10. 2026 (cit. on pp. 85, 88, 111).

[168] CORTADELLA, Jordi; KISHINEVSKY, Mike and GRUNDMANN, Bill: "Synthesis of Synchronous Elastic Architectures". In: *Proceedings of the 43rd Annual Conference on Design Automation - DAC '06*. San Francisco, CA, USA: ACM Press, 2006, p. 657. DOI: 10.1145/1146909.1147077. (Visited on 05/24/2026) (cit. on p. 87).

[169] AMD: ModelSim HDL simulator. https://eda.sw.siemens.com/en-US/ic/modelsim/. Version 2023.4, accessed 2025-05-13. 2026 (cit. on pp. 88, 126, 130).

[170] ANDRONIC, Marta and CONSTANTINIDES, George A.: "NeuraLUT: Hiding Neural Network Density in Boolean Synthesizable Functions". In: *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. Torino, Italy: IEEE, Sept. 2024, pp. 140–148. DOI: 10.1109/FPL64840.2024.00028. (Visited on 05/24/2026) (cit. on p. 92).

[171] SUN, Chang; QUE, Zhiqiang; ZADEH, Bakhtiar; LIU, Qibin; ALVAREZ, Kevin H.; LUK, Wayne and SPIROPULU, Maria: HGQ-LUT: Fast LUT-Aware Training and Efficient Architectures for DNN Inference. 2026. DOI: 10.48550/ARXIV.2604.22293. (Visited on 05/24/2026) (cit. on pp. 92, 169).

[172] KINDGREN, Olof: Invited Paper: A Scalable Approach to IPManagement with FuseSoC. Presented at the Workshop on Open Source Design Automation (OSDA), 2019. 2019. URL: https://osda.gitlab.io/19/kindgren.pdf (cit. on pp. 101, 105, 106, 110).

[173] KINDGREN, Olof: FuseSoC. https://github.com/olofk/fusesoc. Accessed: 2025-05-13. 2015 (cit. on pp. 101, 105, 106, 110).

[174] BAPTIST, Frank Michael: "Training and Optimization of a Graph Neural Network for Deployment on FPGA Hardware in the Belle II Level 1 Trigger". MA thesis. Karlsruhe Institute of Technology, 2026 (cit. on pp. 112, 114).

[175] BHATTACHARYYA, Shuvra S.; MURTHY, Praveen K. and LEE, Edward A.: Software Synthesis from Dataflow Graphs. Vol. 360. The Kluwer International Series in Engineering and

Computer Science. Norwell, MA: Kluwer Academic Publishers, 1996. DOI: 10.1007/978-1-4613-1389-2 (cit. on p. 116).

[176] GUO, Licheng; CHI, Yuze; WANG, Jie; LAU, Jason; QIAO, Weikang; USTUN, Ecenur; ZHANG, Zhiru and CONG, Jason: "AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs". In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 81–92. DOI: 10.1145/3431920.3439289. (Visited on 05/26/2026) (cit. on p. 117).

[177] MYKYTA, Alex: PeakRDL: SystemRDL Register Automation Toolchain. https://github.com/SystemRDL/PeakRDL. Accessed: 2026-04-18. 2026 (cit. on p. 119).

[178] SHANKAR, Murali; DAVIDSAVER, Michael; KONRAD, Martin and LI, LuoFeng: "The EPICS Archiver Appliance". In: *Proceedings of the 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems* ICALEPCS2015 (2015). Ed. by LOU (ED.), Corvetti; KATHLEEN (ED.), Riches and RW (ED.) Volker, Schaa, 4 pages, 0.753 MB. DOI: 10.18429/JACOW-ICALEPCS2015-WEPGF030. (Visited on 04/18/2026) (cit. on p. 121).

[179] HODGSON, Stuart et al.: cocotb: Python-based chip (RTL) verification. https://github.com/cocotb/cocotb. Accessed: 2025-05-13. 2025 (cit. on p. 126).

[180] FORENCICH, Alex: cocotb-axi: Python-based chip (RTL) verification. https://github.com/alexforencich/cocotbext-axi. Accessed: 2025-05-13. 2023 (cit. on p. 126).

[181] NEU, Marc: The DSP Inference Bug That Broke Our Neural Network. https://marcneu.github.io/2025/12/14/vitis-hls-dsp-bug/. Dec. 2025. (Visited on 07/13/2026) (cit. on p. 129).

[182] LOBMAIER, Thomas: "Design and Evaluation of a GNN Algorithm for the ECL L1 Trigger Upgrade at Belle II". MA thesis. Karlsruhe Institute of Technology, Mar. 2026 (cit. on pp. 144–146).

[183] COELHO, Claudionor N. et al.: "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors". en. In: *Nature Machine Intelligence* 3.8 (Aug. 2021), pp. 675–686. DOI: 10.1038/s42256-021-00356-5. URL: https://www.nature.com/articles/s42256-021-00356-5 (cit. on pp. 146, 155).

[184] DE MICHELI, Giovanni: Synthesis and Optimization of Digital Circuits. McGraw-Hill Series in Electrical and Computer Engineering. New York: McGraw-Hill, 1994 (cit. on p. 149).

[185] AMD: Vitis Libraries. https://github.com/Xilinx/Vitis_Libraries. Version 2024.2, accessed 2026-04-29. 2026 (cit. on pp. 152, 153, 205).

[186] NVIDIA: TensorRT. https://github.com/NVIDIA/TensorRT. accessed 2025-06-07. 2025 (cit. on p. 157).

[187] AMD: Zentorch. https://github.com/amd/ZenDNN-pytorch-plugin. accessed 2025-06-07. 2025 (cit. on p. 157).

[188] KIESELER, Jan and QASIM, Shah: Fast Graph Compute. 2024. URL: %5Curl%7B%7Bhttps://github.com/jkiesele/FastGraphCompute%7D%7D (cit. on p. 157).

[189] KALAMKAR, Dhiraj et al.: A Study of BFLOAT16 for Deep Learning Training. 2019. DOI: 10.48550/ARXIV.1905.12322. (Visited on 04/29/2026) (cit. on p. 157).

[190] SUN, Chang; QUE, Zhiqiang; AARRESTAD, Thea; LONCAR, Vladimir; NGADIUBA, Jennifer; LUK, Wayne and SPIROPULU, Maria: "HGQ: High Granularity Quantization for Real-time Neural

Networks on FPGAs". In: *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Seaside CA USA: ACM, Feb. 2026, pp. 79–91. DOI: 10.1145/3748173.3779200. (Visited on 06/06/2026) (cit. on p. 169).

[191] BOGGIA, Laura et al.: Review of Machine Learning for Real-Time Analysis at the Large Hadron Collider Experiments ALICE, ATLAS, CMS and LHCb. 2025. DOI: 10.48550/ARXIV.2506.14578. (Visited on 02/18/2026).

[192] BRUERS, Ben et al.: "Resource-Aware Research on Universe and Matter: Call-to-Action in Digital Transformation". In: *The European Physical Journal Special Topics* (Dec. 2024). DOI: 10.1140/epjs/s11734-024-01436-4. (Visited on 02/18/2026).

[193] HARBAUM, Tanja; SEBOUI, Mahmoud; BALZER, Matthias; BECKER, Jurgen and WEBER, Marc: "A Content Adapted FPGA Memory Architecture with Pattern Recognition Capability for L1 Track Triggering in the LHC Environment". In: *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Washington, DC, USA: IEEE, May 2016, pp. 184–191. DOI: 10.1109/FCCM.2016.52. (Visited on 05/06/2026).

[194] ROSSI, Marco and VALLECORSA, Sofia: "Deep Learning Strategies for ProtoDUNE Raw Data Denoising". In: *Computing and Software for Big Science* 6.1 (Jan. 2022). DOI: 10.1007/s41781-021-00077-9.

Publications

This section contains a complete list of own publications. Publications [Neu23, Neu24a, Neu24b, Neu25b, Neu26a, Neu26b, Neu26c, Neu26d] authored by this thesis' author are listed first. Publication [Neu23] focuses on the development of a MIMO testbed for 6G applications. It does not directly relate to the content of this thesis. The list continues with publications to which the author of this thesis has contributed.

[Neu23] Neu, Marc; Karle, Christian; Nuss, Benjamin; Groeschel, Patrick and Becker, Juergen: "A Scalable and Cost-Efficient Antenna Testbed Using FPGA-Server Compound Structures for Prototyping 6G Applications". In: *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. https://doi.org/10.1109/DCOSS-IoT58021.2023.00039. Pafos, Cyprus: IEEE, 2023. DOI: 10.1109/DCOSS-IoT58021.2023.00039.

[Neu24a] Neu, Marc; Becker, Jürgen; Dorwarth, Philipp; Ferber, Torben; Reuter, Lea; Stefkova, Slavomira and Unger, Kai: "Real-Time Graph Building on FPGAs for Machine Learning Trigger Applications in Particle Physics". In: *Computing and Software for Big Science* (2024). https://doi.org/10.1007/s41781-024-00117-0. DOI: 10.1007/s41781-024-00117-0 (cit. on pp. 49, 50, 52, 71, 74–76, 78, 79, 91, 171).

[Neu24b] Neu, Marc; Karle, Christian; Schmidt, Patrick; Höfer, Julian; Harbaum, Tanja and Becker, Jürgen: "A Dynamically Pipelined Dataflow Architecture for Graph Convolutions in Real-Time Event Interpretation". In: *2024 IEEE 37th International System-on-Chip Conference (SOCC)*. https://doi.org/10.1109/socc62300.2024.10737798. Dresden, Germany: IEEE, 2024. DOI: 10.1109/socc62300.2024.10737798 (cit. on pp. 59, 60).

[Neu25b] Neu, Marc; Haide, Isabel; Justinger, Timo; Rädler, Till; Dajaku, Valdrin; Ferber, Torben and Becker, Jürgen: "Real-Time Graph-based Point Cloud Networks on FPGAs via Stall-Free Deep Pipelining". In: *2025 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*. https://doi.org/10.1109/SBCCI66862.2025.11218652. Manaus, Brazil: IEEE, 2025. DOI: 10.1109/SBCCI66862.2025.11218652. (Visited on 04/29/2026) (cit. on pp. 66–70, 143, 158, 171).

[Neu26a] Haide, I.; Neu, M. et al.: "Real-time graph neural networks on FPGAs for the Belle II electromagnetic calorimeter". In: *Journal of Instrumentation* (2026). https://doi.org/10.1088/1748-0221/21/06/P06039. DOI: 10.1088/1748-0221/21/06/P06039. URL: https://doi.org/10.1088/1748-0221/21/06/P06039 (cit. on pp. 64–67, 93–95, 99, 101, 108, 109, 125, 130, 134, 139, 141, 171, 207).

[Neu26b] Neu, Marc; Baptist, Frank; Lobmaier, Thomas; Papagno, Fabio; Ferber, Torben and Becker, Jürgen: "Reconfigurable Computing Challenge: Real-Time Graph Neural Networks for Online Event Selection in Big Science". In: *2026 IEEE 34th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. https://doi.org/10.1109/FCCM68464.2026.00074. Atlanta, GA, USA: IEEE, 2026. doi: 10.1109/FCCM68464.2026.00074. (Visited on 07/14/2026) (cit. on pp. 143, 148, 155, 171).

[Neu26c] Neu, Marc; Haide, Isabel; Ferber, Torben and Becker, Jürgen: "Real-Time Stream Compaction for Sparse Machine Learning on FPGAs". In: *Submitted to ACAT*. https://doi.org/10.48550/ARXIV.2602.23281. arXiv, 2026. doi: 10.48550/ARXIV.2602.23281. (Visited on 04/29/2026) (cit. on pp. 46, 171).

[Neu26d] Neu, Marc et al.: "Commissioning and Low Latency Operation of the Graph Neural Network Electromagnetic Calorimeter Trigger at the Belle II Experiment". In: *Submitted to IEEE TNS*. https://doi.org/10.48550/ARXIV.2607.09347. arXiv, 2026. doi: 10.48550/ARXIV.2607.09347. (Visited on 07/13/2026) (cit. on pp. 93, 104, 118, 132, 134, 135, 138, 140).

[Pfa23] Pfau, Johannes; Leys, Richard; Neu, Marc; Serdyuk, Alexey; Peric, Ivan and Becker, Jürgen: "A Unified SoC Lab Course: Combined Teaching of Mixed Signal Aspects, System Integration, Software Development and Documentation". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. https://doi.org/10.1109/ISCAS46773.2023.10181679. IEEE, 2023. doi: 10.1109/ISCAS46773.2023.10181679.

[Kar23] Karle, Christian; Neu, Marc; Pfau, Johannes; Sperling, Jan and Becker, Jürgen: "ReLoDAQ: Resource-Efficient, Low-Overhead 200 $Gbits^{-1}$ Data Acquisition System for 6G Prototyping". In: *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. https://doi.org/10.1109/FCCM57271:2023.00037. IEEE, 2023. doi: 10.1109/FCCM57271:2023.00037.

[Ung23a] Unger, Kai; Becker, Jürgen; Kiesling, Christian; Ma, Yichuan; Meggendorfer, Felix; Neu, Marc; Schmidt, Elia and Zweigart, Ulrike: "A Convolution Neural Network Based Displaced Vertex Trigger for the Belle II Experiment". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. https://doi.org/10.1007/978-3-031-42921-7_12. Cham: Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-42921-7_12 (cit. on p. 15).

[Ung23b] Unger, Kai; Neu, Marc; Becker, Jürgen; Schmidt, Elias; Kiesling, Christian; Meggendorfer, Felix and Skambraks, Sebastian: "Data-Driven Design of the Belle II Track Segment Finder". In: *Journal of Instrumentation* (2023). https://doi.org/10.1088/1748-0221/18/02/c02001. doi: 10.1088/1748-0221/18/02/c02001 (cit. on pp. 15, 75).

[Kar24a] Karle, Christian; Neu, Marc; Nuss, Benjamin; Witte, Lukas; Scheder, Andre; Waldner, Eva; Shkurtaj, Ema; Harbaum, Tanja and Becker, Jürgen: "Scalable Multi-Level Synchronization Technique of Distributed Multi-RFSoC-Server Systems for 6G". In: *2024 IEEE 37th International System-on-Chip Conference (SOCC)*. https://doi.org/10.1109/socc62300.2024.10737777. Dresden, Germany: IEEE, 2024. doi: 10.1109/socc62300.2024.10737777.

[Kar24b] Karle, Christian; Neu, Marc; Nuss, Benjamin; Chen, Jiayi; Witte, Lukas; Scheder, Andre; Harbaum, Tanja and Becker, Jürgen: "Modular Hardware Design for High-Performance MIMO-Capable SDR Systems to Accelerate 6G Development". In: *2024 IEEE 37th International System-on-Chip Conference (SOCC)*. https://doi.org/10.1109/socc62300.2024.10737765. Dresden, Germany: IEEE, 2024. doi: 10.1109/socc62300.2024.10737765.

[Nus24] Nuss, Benjamin; Karle, Christian; Neu, Marc; Witte, Lukas; Scheder, Andre; Franchi, Norman; Vossiek, Martin; Becker, Juergen and Zwick, Thomas: "Flexible and Scalable Broadband Massive MIMO Testbed for Joint Communication and Sensing Applications". In: *European Wireless 2024 : 29th European Wireless Conference*. Brno, Czech Republic: VDE-Verlag, 2024.

[Sch24] Scheder, Andre; Witte, Lukas; Rottinghaus, Jonas; Gröschel, Patrick; Karle, Christian; Neu, Marc; Nuss, Benjamin and Vossiek, Martin: "An Advanced Concept for Coherent Ultra-Low Phase-Noise Clock, LO and Trigger Generation & Distribution in 6G Massive-MIMO Systems". In: *2024 54th European Microwave Conference (EuMC)*. https://doi.org/10.23919/eumc61614.2024.10732511. Paris, France: IEEE, 2024. doi: 10.23919/eumc61614.2024.10732511.

[Ung25] Unger, Kai Lukas; Becker, Jürgen; Forsthofer, Timo; Hiesl, Simon; Kiesling, Christian; Neu, Marc; Schmidt, Elia; Meggendorfer, Felix; Mi, Tiancheng and Ou, Zuwen: "A Multi-Hough-based Displaced Vertex Track Trigger for the Belle II Experiment". In: *Journal of Instrumentation* (2025). https://doi.org/10.1088/1748-0221/20/02/c02051. doi: 10.1088/1748-0221/20/02/c02051 (cit. on p. 15).

[Hei26b] Heine, Greta; Mayer, Fabio; Neu, Marc; Becker, Jürgen and Ferber, Torben: "Hardware-Aware Design of a GNN-Based Hit Filtering Algorithm for the Belle II Level-1 Trigger". In: *Submitted to ACAT*. https://doi.org/10.48550/ARXIV.2602.17761. arXiv, 2026. doi: 10.48550/ARXIV.2602.17761. (Visited on 05/06/2026) (cit. on pp. 4, 71, 72).

[Kar25] Karle, Christian et al.: "Programmable RFSoC-Based Antenna Emulator for MIMO System Prototyping". In: *2025 IEEE 38th International System-on-Chip Conference (SOCC)*. https://doi.org/10.1109/SOCC66126.2025.11235346. Dubai, United Arab Emirates: IEEE, 2025. doi: 10.1109/SOCC66126.2025.11235346. (Visited on 07/04/2026).

[Lai25] Lai, Y.-T. et al.: "Design of the Global Reconstruction Logic in the Belle II Level-1 Trigger System". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2025). https://doi.org/10.1016/j.nima.2025.170577. doi: 10.1016/j.nima.2025.170577. (Visited on 01/05/2026) (cit. on pp. 2, 11, 12, 71).

[Bäh25] Bähr, Steffen et al.: "The Neural Network First-Level Hardware Track Trigger of the Belle II Experiment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2025). https://doi.org/10.1016/j.nima.2025.170279. doi: 10.1016/j.nima.2025.170279. (Visited on 01/05/2026) (cit. on pp. 15, 42, 71).

[Hei26c] Heine, Greta; Mayer, Fabio; Neu, Marc; Becker, Jürgen and Ferber, Torben: "Hardware-Accelerated GNN-based Hit Filtering for the Belle II Level-1 Trigger". In: *Journal of Instrumentation* (2026). https://doi.org/10.1088/1748-0221/21/02/C02007. DOI: 10.1088/1748-0221/21/02/C02007. (Visited on 02/21/2026) (cit. on pp. 4, 71, 72).

[Sot26] Sotiropoulos, Georgios; Neu, Marc; Harbaum, Tanja; Ferber, Torben and Becker, Jürgen: "RTL Fault Injection of a Deployed Graph Neural Network Trigger for Belle II". In: *2026 IEEE 39th International System-on-Chip Conference (SOCC)*. Heidelberg, Germany: IEEE, 2026.

[Par26] Paraskevas, Foivos; Neu, Marc; Harbaum, Tanja; Sezer, Sakir and Becker, Jürgen: "Integration of a Row-Stationary CNN Accelerator and a RISC-V Core: Clock-Ratio and Throughput Analysis". In: *2026 IEEE 39th International System-on-Chip Conference (SOCC)*. Heidelberg, Germany: IEEE, 2026.

Student Theses

This section contains a list of student theses supervised by this thesis' author. Entries are sorted by year and author name. For [Zwe22, Spe23b, Yic23, Jin24], the author of this work was the secondary supervisor.

[Keh22] Kehrberger, Lennart: "Entwurf eines webbasierten Speichermana-gements für DAQ-Systeme im Rahmen der 6G-Mobilfunkentwicklung". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Zwe22] Zweigart, Ulrike: "Design of a Convolutional Neural Network for displaced vertex traces in the Belle II experiment". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Bus23] Busch, Karl: "Hardwarenahe Konzeption eines Graph Neural Networks für 3D Objekterkennung". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Sha23] Shao, Shen: "Parallelization of Graph Neural Networks by using compiler-based high-level synthesis tools". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Spe23a] Sperling, Jan: "Edge Detection for Graph Construction under Locality Constraints for the Belle II Experiment". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Spe23b] Sperling, Jan: "FPGA-basierte Datenstromsynchronisierung für 6G-Kommunikationsanwendungen". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Yic23] Yichuan, Ma: "Design and evaluation of a CNN based displaced vertex trigger for the Belle II experiment". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Jin24] Jingjang, He: "Compiler-Based Integration of Neural Network Accelerator". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2024.

[Daj25] Dajaku, Valdrin: "Hardware-Aware Co-Design of Dynamic Graph Neural Networks for the Belle II Experiment". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025.

[Fre25] Frey, Yoshua: "Performance Comparision of HLS Compilers in Low-Latency High-Throughput Applications". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025 (cit. on p. 38).

[Jus25] Justinger, Timo: "Hierarchisches Floorplanning von Datenflussarchitekturen auf FPGAs im Belle II Experiment". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025.

[May25] Mayer, Fabio: “Real-Time Inference of Static Graph Neural Networks on FPGAs for the Belle II Detector”. Master’s Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025 (cit. on pp. 72, 82, 92).

[Pap25a] Papagno, Fabio: “Triggerarchitekturen für Echtzeitclustering im Rahmen des Belle II Calorime-ter Upgrades”. Master’s Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025.

[Räd25] Rädler, Till: “Skalierbare Graph Neuronale Netzwerke durch CGRAs für den Belle2 Detektor”. Master’s Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2025.

Figures

Tables

This page intentionally left blank

Listings

This page intentionally left blank

Algorithms

This page intentionally left blank

Acronyms

ADC	Analog Digital Converter
AIE	AI Engine
ALU	Algorithmic Logic Unit
ANN	All Nearest Neighbour
API	Application Programming Interface
ARICH	Aerogel RICH Detector
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
B2TT	Belle II Trigger Timing
BRAM	Block Random-Access Memory
CDC	Central Drift Chamber
CERN	European Organization for Nuclear Research
CGRA	Coarse Grain Reconfigurable Array
CLB	Configurable Logic Block
CMS	Compact Muon Solenoid
CNN	Convolutional Neural Network
COOPER	COmmon Pipelined Platform for Electronics Readout
CPCS	Condensation Point Candidate Selection
CPS	Condensation Point Selection
CPU	Central Processing Unit
CsI(Tl)	thallium-doped cesium iodide
DAQ	Data Acquisition
DDR-RAM	Double Data Rate Random Access Memory
DMA	Direct Memory Transfer
DNN	Deep Neural Network
DSP	Digital Signal Processing

ECL	Electromagnetic Calorimeter
FAM	Flash ADC Analog Module
FF	Flip-Flop Register
FIFO	first-in, first-out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDL	Global Decision Logic
GNN	Graph Neural Network
GNN-ETM	Graph Neural Network Electromagnetic Calorimeter Trigger Module
GNN-HCM	Graph Neural Network Hit Cleanup Trigger Module
GPIO	general purpose input output
GPU	Graphics Processing Unit
GRL	Global Reconstruction Logic
GT	gigabit transreceiver
HAL	Hardware Abstraction Layer
HLS	High Level Synthesis
HLT	High-Level Trigger System
IC	Integrated Circuit
ICN-ETM	Isolated Cluster Network Electromagnetic Calorimeter Trigger Module
IO	input output
IP	Intellectual Property
KLM	K_L^0 / Muon Detector
L1 Trigger	First-Level Trigger System
LHC	Large Hadron Collider
LUT	Lookup Table
MLIR	Multi-Level Intermediate Representation
MLP	Multi Layer Perceptron
NoC	Network-on-Chip

NPU	Neural Processing Unit
NSM 2	Network Shared Memory 2
ONNX	Open Neural Network Exchange
PCN	Point Cloud Network
PE	Processing Element
PIN	p-type, intrinsic, n-type
PV	Process Variable
PXD	Pixel Detector
RAM	Random-Access Memory
RTL	Register Transfer Level
Shaper-DSP	Shaper-Digital Signal Processor and Collector Module
SLR	Super Logic Region
SoC	System-on-Chip
SSI	Silicon Stacked Interposer
SVD	Silicon Vertex Detector
TC	Trigger Cell
TMM	Trigger Merger Module
TOP	Time-Of-Propagation Counter
UT	Universal Trigger Board
VLIW	Very Large Instruction Word
VME	Versa Module Eurocard
XRT	Xilinx Runtime

This page intentionally left blank

Glossary

AMD Vitis	Commercial high-level synthesis and software development kit by AMD.
AMD Vivado	Commercial digital synthesis, implementation, and simulation tool by AMD.
BRAM	Block Random Access Memory on FPGAs. Typically implemented in SRAM. Higher density than distributed RAM, but lower density than external memory.
CDC	Central Drift Chamber, a subdetector of the Belle II Experiment.
CLB	Logic cluster primitive in AMD FPGAs.
data window	Sampling period in which all sensors, or channels, of a subdetector are read out simultaneously.
DeePloy	Deployment methodology for mapping graph-based Point Cloud Networks onto Field Programmable Gate Arrays and Coarse Grained Reconfigurable Arrays developed in this thesis.
DSP	Hardened Arithmetic Logic Unit such as multipliers on FPGAs.
ECL	Electromagnetic Calorimeter, a subdetector of the Belle II Experiment.
event	Aggregation of data windows from all subdetectors for a given timeframe. Generated by the event builder in the high-level trigger at the Belle II Experiment. Typically used for offline analysis.
FF	Atomic storage element of digital circuits which allow to store a persistent state.
LUT	Lookup Table, a combination of memory and multiplexer that allows deriving output values from a table according to an input value.

ModelSim Commercial register-transfer level simulation tool by Siemens.

operation A formalized algorithmic transformation, for example a linear transformation or a ReLU used to describe neural network architectures. In comparison to operators, operations are more complex and usually at a higher level of abstraction as their definition includes assumptions about the memory layout and access patterns.

operator A primitive, atomic computation such as addition, subtraction or multiplication. In contrast to an operation, an operator carries no assumptions about memory layout or access patterns.

PE An abstract representation of a data-driven module implemented in hardware.

TC Analog sum of 4×4 crystals in the ECL subdetector of the Belle II Experiment. Used as input of the clustering algorithm at trigger level.

trigger bit Boolean flag generated by hardware modules in the Belle II trigger system. Determines, if a trigger window should be stored as offline event for further analysis.

trigger window Sampling period which is used in first-level trigger system at the Belle II Experiment. Typically represents a sliding window over one or more data windows. The precision of readout channels is usually reduced for the trigger readout.

Appendix A

CaloClusterNet Hyperparameters

A.1 Radiant Armadillo

This section contains additional information for the Radiant Armadillo CaloClusterNet model which is used in the GNN-ETM. An overview of the model is given in figure 6.10. The following tables detail the model-dependend hyperparameters: Table A.1 lists the trainable layers and their hyperparameters. Table A.2 lists the toplogy layers wise and their hyperparameters. Table A.3 details the configuration of the condensation point selection. Table A.4 details the configuration of the condensation point selection. D_i notes the input dimension, D_o denotes the output dimension of a given layer.

Table A.1: Model-dependend hyperparameters for trainable layers in the Radiant Armadillo CaloClusterNet model.

ID	Act. Type	D_i	D_o	Input Type	Output Type	Weight Type	Comment
0	ReLU	5	16	Q4.10	Q3.5	Q4.5	–
1	ReLU	5	5	Q4.10	Q3.5	Q3.5	–
2	Linear	16	8	Q3.5	Q3.13	Q3.13	Feature Space
3	Linear	16	6	Q3.5	Q3.13	Q3.13	Latent Space
6	ReLU	32	32	Q3.13	Q3.13	Q4.13	–
7	ReLU	32	16	Q3.13	Q3.5	Q3.5	–
8	ReLU	16	16	Q3.5	Q3.5	Q4.5	–
9	Linear	16	6	Q3.13	Q3.13	Q3.13	Feature Space
10	Linear	16	8	Q3.13	Q3.13	Q3.13	Latent Space
13	ReLU	32	32	Q3.13	Q3.13	Q4.13	–
14	ReLU	16	16	Q3.13	Q3.5	Q3.5	–
16	ReLU	16	16	Q3.5	Q3.5	Q3.5	–
17	ReLU	16	1	Q3.5	Q5.11	Q6.10	Energy Scale Factor
18	Sigmoid	16	1	Q3.5	Q3.13	Q6.10	Signal
19	Linear	16	3	Q3.5	Q5.11	Q6.10	Position
20	Linear	16	3	Q3.5	Q5.11	Q6.10	CCords
21	Sigmoid	16	1	Q3.5	Q3.13	Q6.10	Beta

Table A.2: Model-dependend hyperparameters for toplogy layers in the Radiant Armadillo CaloClusterNet model.

ID	Type	D_i	D_o	Input Type	Output Type	Comment
5	Concat	(16,16)	32	(Q3.5, Q3.5)	Q3.5	–
12	Concat	(16,16)	32	(Q3.5, Q3.5)	Q3.5	–
15	Concat	(16,16,5)	37	(Q3.5, Q3.5, Q3.5)	Q3.5	–
23	Mult	1	1	Q6.10	Q6.10	Elementwise Multiply

Table A.3: Model-dependend hyperparameters for GraVNetConv layers in the Radiant Armadillo CaloClusterNet model.

Parameter	Value
Coordinate Type	Q3.13
Feature Type	Q3.13
Distance Type	Q7.13
Exponential Type	UQ1.9
Output Type	Q3.13
Coordinate Space Dimensionality	6
Feature Space Dimenstionality	8
K	8

Table A.4: Model-dependend hyperparameters for Condensation Point Selection layers in the Radiant Armadillo CaloClusterNet model.

Parameter	Value
Input Type	Q6.10
Distance Type	Q12.8
Beta Type	Q6.10
Beta dimensionality	1
osition dimensionality	3

A.2 Sunset

This section contains additional information for the Sunset CaloClusterNet model which is used in the GNN-ETM. An overview of the model is given in figure 6.11. The following tables detail the model-dependend hyperparameters: Table A.5 lists the trainable layers and their hyperparameters. Table A.6 lists the toplogy layers wise and their hyperparameters. Table A.7 details the configuration of the condensation point selection. Table A.8 details the configuration of the condensation point selection. D_i notes the input dimension, D_o denotes the output dimension of a given layer.

Table A.5: Model-dependend hyperparameters for trainable layers in the Sunset CaloClusterNet model.

ID	Act. Type	D_i	D_o	Input Type	Output Type	Weight Type	Comment
0	ReLU	5	16	Q4.10	Q2.6	Q1.7	–
1	ReLU	5	5	Q4.10	Q2.6	Q1.7	–
2	Linear	16	8	Q2.6	Q2.6	Q2.6	Feature Space
3	Linear	16	6	Q2.6	Q2.6	Q2.6	Latent Space
6	ReLU	32	32	Q2.6	Q2.6	Q1.7	–
7	ReLU	16	16	Q2.6	Q2.6	Q1.7	–
9	ReLU	16	16	Q2.6	Q2.6	Q1.7	–
17	ReLU	16	1	Q2.6	Q6.10	Q2.6	Energy Scale Factor
18	Sigmoid	16	1	Q2.6	Q6.10	Q4.4	Signal
19	Linear	16	3	Q2.6	Q6.10	Q2.6	Position
20	Linear	16	3	Q2.6	Q6.10	Q2.6	CCords
21	Sigmoid	16	1	Q2.6	Q6.10	Q4.4	Beta

Table A.6: Model-dependend hyperparameters for toplogy layers in the Sunset CaloClusterNet model.

ID	Type	D_i	D_o	Input Type	Output Type	Comment
5	Concat	(16,16)	32	(Q2.6, Q2.6)	Q2.6	–
8	Concat	(16,5)	21	(Q2.6, Q2.6)	Q2.6	–
23	Mult	1	1	Q6.10	Q6.10	Elementwise Multiply

Table A.7: Model-dependend hyperparameters for GraVNetConv layers in the Sunset CaloClusterNet model.

Parameter	Value
Coordinate Type	Q2.6
Feature Type	Q2.6
Distance Type	Q10.10
Exponential Type	UQ1.9
Output Type	Q2.6
Coordinate Space Dimensionality	6
Feature Space Dimensionality	8
K	8

Table A.8: Model-dependend hyperparameters for the Condensation Point Selection layers in the Sunset CaloClusterNet model.

Parameter	Value
Input Type	Q6.10
Distance Type	Q12.8
Beta Type	Q6.10
Beta Dimensionality	1
Position Dimensionality	3

A.3 Long Shutdown 2

This section contains additional information for the CaloClusterNet model which is used in appendix 7 for the study of the Belle II upgrade during long shutdown 2. An overview of the model is given in figure 7.2. The following tables detail the model-dependend hyperparameters: Table A.9 lists the trainable layers and their hyperparameters. Table A.10 lists the toplogy layers wise and their hyperparameters. Table A.11 details the configuration of the condensation point selection. Table A.12 details the configuration of the condensation point selection. D_i notes the input dimension, D_o denotes the output dimension of a given layer.

Table A.9: Model-dependend hyperparameters for trainable layers in the Long Shutdown 2 CaloClusterNet model.

ID	Act. Type	D_i	D_o	Input Type	Output Type	Weight Type	Comment
0	ReLU	5	8	Q3.12	UQ1.7	Q0.7	–
1	ReLU	5	8	Q3.12	UQ1.7	Q0.7	–
2	ReLU	8	16	UQ1.7	UQ1.7	Q0.7	–
3	Linear	16	8	UQ1.7	Q1.6	Q0.7	Feature Space
4	Linear	16	6	UQ1.7	Q1.6	Q0.7	Latent Space
7	ReLU	32	32	Q1.6	UQ1.7	Q0.7	–
8	ReLU	32	16	UQ1.7	UQ1.7	Q0.7	–
9	ReLU	16	16	UQ1.7	UQ1.7	Q0.7	–
10	Linear	16	6	UQ1.7	Q1.6	Q0.7	Feature Space
11	Linear	32	8	UQ1.7	Q1.6	Q0.7	Latent Space
14	ReLU	32	32	Q1.6	UQ1.7	Q0.7	–
15	ReLU	16	16	UQ1.7	UQ1.7	Q0.7	–
17	ReLU	16	16	UQ1.7	UQ1.7	Q0.7	–
18	ReLU	16	1	UQ1.7	UQ4.11	Q0.7	Energy Scale Factor
19	Sigmoid	16	1	UQ1.7	Q4.11	Q6.10	Signal
20	Linear	16	3	UQ1.7	Q4.11	Q6.10	Position
21	Linear	16	3	UQ1.7	Q4.11	Q6.10	CCords
22	Sigmoid	16	1	UQ1.7	Q5.10	Q2.13	Beta

Table A.10: Model-dependend hyperparameters for toplogy layers in the Long Shutdown 2 CaloClusterNet model.

ID	Type	D_i	D_o	Input Type	Output Type	Comment
6	Concat	(16,16)	32	(UQ1.7, Q1.6)	Q1.6	–
13	Concat	(16,16)	32	(UQ1.7, Q1.6)	Q1.6	–
16	Concat	(16,16,5)	37	(UQ1.7, UQ1.7, UQ1.7)	UQ1.7	–
24	Mult	1	1	UQ4.11	UQ4.11	Elementwise

Table A.11: Model-dependend hyperparameters for GraVNetConv layers in the Long Shutdown 2 CaloClusterNet model.

Parameter	Value
Coordinate Type	Q1.6
Feature Type	Q1.6
Distance Type	Q5.6
Exponential Type	UQ1.7
Output Type	Q1.6
Coordinate Space Dimensionality	6
Feature Space Dimenstionality	8
K	8

Table A.12: Model-dependend hyperparameters for Condensation Point Selection layers in the Long Shutdown 2 CaloClusterNet model.

Parameter	Value
Input Type	Q4.11
Distance Type	Q8.10
Beta Type	Q4.11
Beta dimensionality	1
osition dimensionality	3

This page intentionally left blank

Appendix B

GNN-ETM Interfaces

B.1 ICN-ETM to GNN-ETM Packet Definition

The interface between the ICN-ETM and GNN-ETM is realised via gigabit transceivers. For each detector snapshot, a packet containing all TCs is transmitted. Each packet consists of 12 288 bit, split into 16 consecutive frames of 768 bit each. The fields of each frame are shown in figure B.1. Each frame starts with a 20 bit header, consisting of the frame clock counter and the FAM Revoclk counter. The frame clock counter determines the position of the frame within the packet, counting from 0 to 15. The FAM Revoclk counter is derived from the local clock counter on the FAM module and can be used to compare the timing of different systems, e.g., the ICN-ETM and the GNN-ETM.

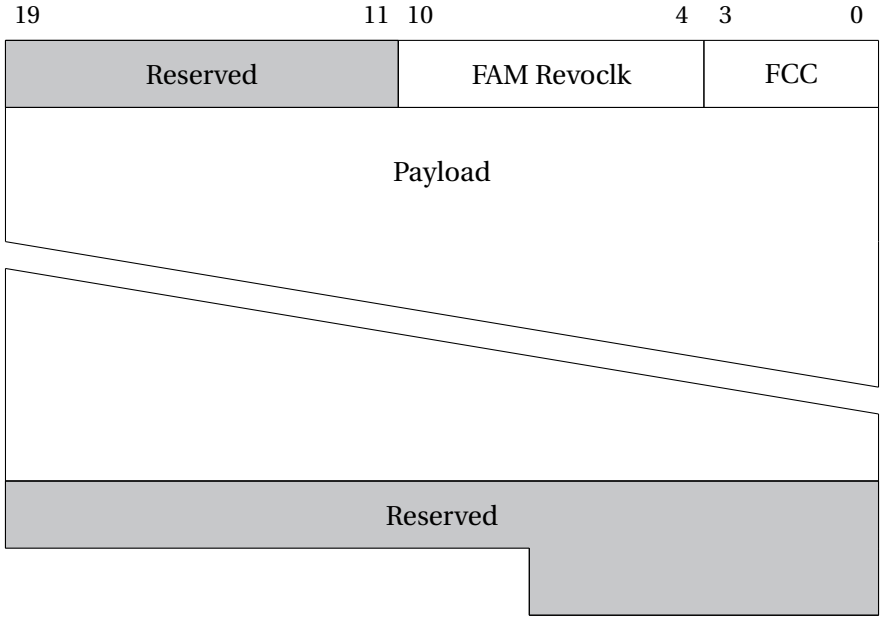


Figure B.1: Frame fields (768 bit). FCC = Frame Clock Counter

The 576 TCs of the ECL are distributed across the 16 frames of a packet. The payload of each frame contains 36 channels of 20 bit each, where each channel carries data for a single TC at a given point in time. The trigger cell identifier is determined by the frame position t within the packet and the channel index c within the frame, as shown in equation (B.1). The bit layout of a channel is shown in figure B.2.

$$\mathrm{TC}(t,c) = 36t + c + 1, \qquad t \in \{0,\dots,15\}, \quad c \in \{0,\dots,35\}. \tag{B.1}$$

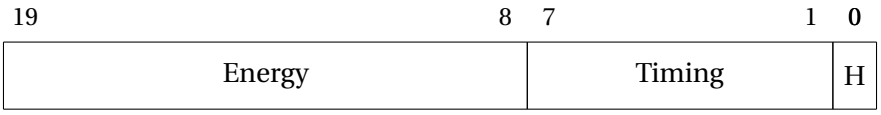


Figure B.2: Channel fields (20 bit). H = hit flag.

B.2 GNN-ETM to GDL Packet Definition

The interface between GNN-ETM and GDL is realised via gigabit transceivers. For each trigger window, e.g. every 125 ns, a packet is sent. The definition of the packet is shown in figure B.3. A valid packet is indicated by the hit flag and indicates that at least one TC has been found in the current trigger window. The time difference between various systems, such as ICN-ETM and GNN-ETM, can be analyzed through the event time. The fine event time has a LSB resolution of 1 ns and is derived from the timing calibration in the preprocessing stage of GNN-ETM. The coarse event time has a LSB resolution of 128 ns and is derived from the FAM revoclk signal. The position of all available trigger bits in given in tables B.1 and B.2.

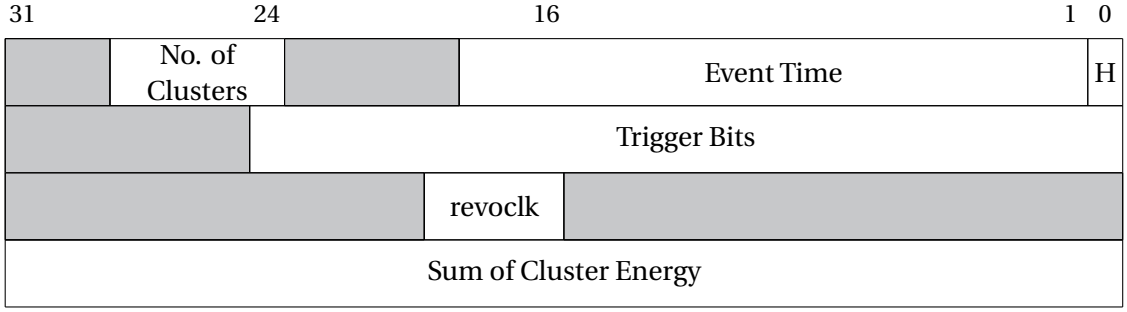


Figure B.3: GNN-ETM output packet fields (128 bit). Fields marked gray are reserved. H = fit flag.

Table B.1: Position of individual trigger bits in the GDL packet.

Trigger Line	Description	Address
`eclgnn_cpt_c2_100`	At least 2 clusters with $E > 100\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x20`
`eclgnn_cpt_c3_100`	At least 3 clusters with $E > 100\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x21`
`eclgnn_cpt_c4_100`	At least 4 clusters with $E > 100\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x22`
`eclgnn_cpt_c2_125`	At least 2 clusters with $E > 125\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x23`
`eclgnn_cpt_c3_125`	At least 3 clusters with $E > 125\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x24`
`eclgnn_cpt_c4_125`	At least 4 clusters with $E > 125\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x25`
`eclgnn_cpt_c2_150`	At least 2 clusters with $E > 150\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x26`
`eclgnn_cpt_c3_150`	At least 3 clusters with $E > 150\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x27`
`eclgnn_cpt_c4_150`	At least 4 clusters with $E > 150\,\text{MeV}$ and $1 < \theta_{\text{id}} < 16$.	`0x28`
`eclgnn_sig_c2_100`	At least 2 clusters with signal classification, $E > 100\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x29`
`eclgnn_sig_c3_100`	At least 3 clusters with signal classification, $E > 100\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2A`
`eclgnn_sig_c4_100`	At least 4 clusters with signal classification, $E > 100\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2B`
`eclgnn_sig_c2_125`	At least 2 clusters with signal classification, $E > 125\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2C`
`eclgnn_sig_c3_125`	At least 3 clusters with signal classification, $E > 125\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2D`
`eclgnn_sig_c4_125`	At least 4 clusters with signal classification, $E > 125\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2E`
`eclgnn_sig_c2_150`	At least 2 clusters with signal classification, $E > 150\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x2F`
`eclgnn_sig_c3_150`	At least 3 clusters with signal classification, $E > 150\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x30`
`eclgnn_sig_c4_150`	At least 4 clusters with signal classification, $E > 150\,\text{MeV}$, and $1 < \theta_{\text{id}} < 16$.	`0x31`

Table B.2: Position of individual trigger bits in the GDL packet (continued).

Trigger Line	Description	Address
`eclgnn_cpt_esum_1000`	Cluster energy sum ≥ 1 GeV and $1 < \theta_{\mathrm{id}} < 16$.	`0x32`
`eclgnn_cpt_esum_1500`	Cluster energy sum ≥ 1.5 GeV and $1 < \theta_{\mathrm{id}} < 16$.	`0x33`
`eclgnn_sig_esum_1000`	Cluster energy sum ≥ 1 GeV with signal classification and $1 < \theta_{\mathrm{id}} < 16$.	`0x34`
`eclgnn_sig_esum_1500`	Cluster energy sum ≥ 1.5 GeV with signal classification and $1 < \theta_{\mathrm{id}} < 16$.	`0x35`
`eclgnn_lml1`	At least 1 cluster with signal classification, $E > 2$ GeV and $3 < \theta_{\mathrm{id}} < 15$.	`0x36`
`eclgnn_lml6`	Exactly 1 cluster with signal classification, $E > 1$ GeV and $3 < \theta_{\mathrm{id}} < 16$; exactly 1 cluster with signal classification and $E > 0.3$ GeV.	`0x37`
`eclgnn_lml7`	Exactly 1 cluster with signal classification, $E > 1$ GeV and $\theta_{\mathrm{id}} \in \{2, 3, 16\}$; exactly 1 cluster with signal classification and $E > 0.3$ GeV (any θ_{id}).	`0x38`

B.3 GNN-ETM to B2Link Packet Definition

The GNN-ETM output packet is transmitted over Belle2Link. It consists of a fixed-size header region followed by two variable-length payloads containing TCs and clusters. Three consecutive trigger windows are stored in one Belle2Link packet. Figure B.4 gives an overview of the complete packet. Multi-byte fields are transmitted in little-endian byte order. Reserved bits are gray.

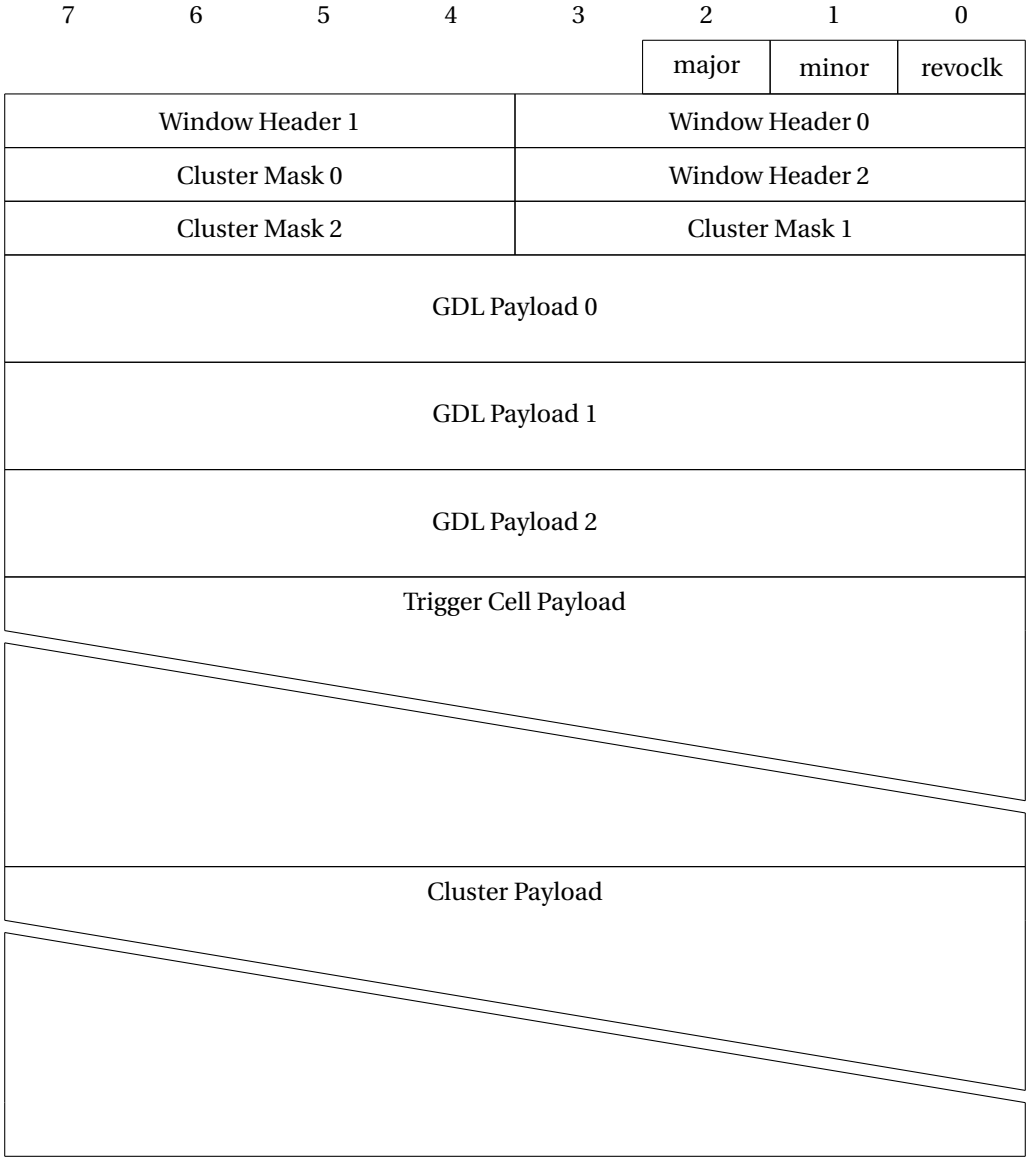


Figure B.4: Overall layout of a GNN-ETM B2Link packet.

Trigger Window Header

Figure B.5 details the trigger window header.

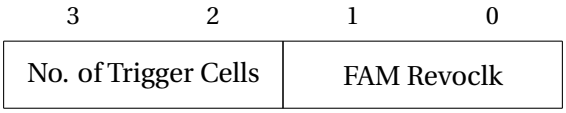


Figure B.5: Byte field for the trigger window header.

GDL Payload

Figure B.6 details the GDL packet payload field.

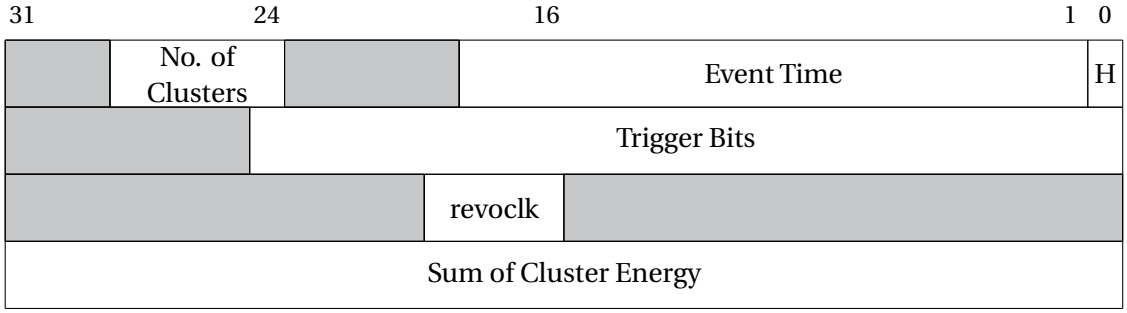


Figure B.6: GDL payload field. H = fit flag.

Trigger Cell Payload

Figure B.7 details the TC payload field.

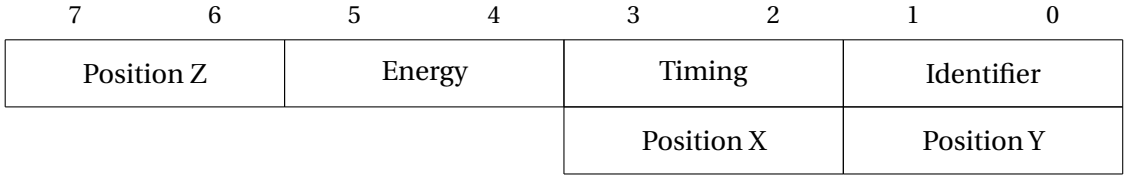


Figure B.7: Structure of a single TC in the TC payload field.

Cluster Payload

Figure B.8 details the cluster payload field.

7 6	5 4	3 2	1 0
Position X	Energy	Beta	Identifier
CCords X	Signal	Position Z	Position Y
		CCords Z	CCords Y

Figure B.8: Structure of a single cluster in the cluster payload field.

This page intentionally left blank

Appendix C

GNN-ETM Register Map

The GNN-ETM firmware exposes its control and monitoring interface through a hierarchical SystemRDL register file. All registers are 32 bit wide and accessed through the VME bus on the UT 4 carrier board. The top-level address map aggregates four sub-maps, each covering a functional domain of the firmware as listed in table C.1. All offsets are given relative to the base address of the containing submap. Remaining bit positions not listed in any field table are reserved for future use.

Table C.1: Top-level address map of the GNN-ETM firmware. Offsets are relative to the VME base address of the module.

Base offset	Submap	Purpose
`0x0000_0000`	`ctrl_vme`	VME identification and clock monitoring.
`0x0000_1000`	`ctrl_global`	Global reset, link status, algorithm thresholds.
`0x0000_1200`	`ctrl_belle2link`	Belle2Link configuration and status.
`0x0000_1400`	`ctrl_trigger`	Trigger monitoring and control.

C.1 VME Registers

The `ctrl_vme` submap provides firmware identification and clock-domain monitoring. Identifier and git-hash registers are populated at synthesis time and are used to verify the firmware version. The four clock counters are each split into a low and a high 32 bit half, together forming a 64 bit free-running counter. All registers carry a single 32 bit `value` field and are summarised in table C.2.

Table C.2: Registers of the `ctrl_vme` submap.

Offset	Register	Reset	Description
`0x10`	`FIRMWARE_ID`	`0x0000_0000`	Firmware identifier in ASCII format.
`0x14`	`GIT_HASH`	`0xDDDD_DDDD`	Git hash of the firmware revision.
`0x40`	`GTHGC_COUNTER_LSB`	`0x0000_0000`	GTH GC clock counter, low word.
`0x44`	`GTHGC_COUNTER_MSB`	`0x0000_0000`	GTH GC clock counter, high word.
`0x48`	`GTYGC_COUNTER_LSB`	`0x0000_0000`	GTY GC clock counter, low word.
`0x4C`	`GTYGC_COUNTER_MSB`	`0x0000_0000`	GTY GC clock counter, high word.
`0x60`	`SYS_COUNTER_LSB`	`0x0000_0000`	System clock counter, low word.
`0x64`	`SYS_COUNTER_MSB`	`0x0000_0000`	System clock counter, high word.
`0x68`	`DATA_COUNTER_LSB`	`0x0000_0000`	Data clock counter, low word.
`0x6C`	`DATA_COUNTER_MSB`	`0x0000_0000`	Data clock counter, high word.

C.2 Global Registers

The `ctrl_global` submap holds the global firmware control register, the status registers of the four gigabit serial links connecting the GNN-ETM to the ICN-ETM, the GDL, and the GRL, as well as the runtime parameters of the CaloClusterNet algorithm. Table C.3 gives an overview.

Table C.3: Registers of the `ctrl_global` submap.

Offset	Register	Description
`0x000`	`CTRL`	Global control and reset register.
`0x004`	`CPS_BETA_THRESHOLD`	Condensation point selection beta threshold.
`0x008`	`CPS_ISOLATION_THRESHOLD`	Condensation point selection isolation threshold.
`0x00C`	`CCN_SIGNAL_THRESHOLD`	CaloClusterNet signal threshold.
`0x040`	`LINK_ICN_RX`	ICN-ETM receive link status.
`0x044`	`LINK_GDL_RX`	GDL receive link status.
`0x048`	`LINK_GDL_TX`	GDL transmit link status.
`0x04C`	`LINK_GRL_RX`	GRL receive link status.
`0x050`	`LINK_GRL_TX`	GRL transmit link status.
`0x1FC`	`B2L_PACKET_VERSION`	Belle2Link packet version.

Global Control

Table C.4 lists the fields of `CTRL`.

Table C.4: Field descriptions of `CTRL` at offset `0x000`. Remaining fields are reserved for future use.

Bit	Field	Type	Reset	Description
10	`reset_link_grl`	R/W	0x0	Resets the GT transceivers on the link between GNN-ETM and GRL.
9	`reset_link_gdl`	R/W	0x0	Resets the GT transceivers on the link between GNN-ETM and GDL.
8	`reset_link_icn`	R/W	0x0	Resets the GT transceivers on the link between ICN-ETM and GNN-ETM.
1	`done`	R	0x0	Indicates that the reset is complete.
0	`reset`	R/W	0x1	Resets the full GNN-ETM trigger pipeline.

ICN-ETM Receive Link Status

Table C.5 lists the fields of `LINK_ICN_RX`.

Table C.5: Field descriptions of `LINK_ICN_RX` at offset `0x040`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
3:0	`lane_up`	R	0x0	Bitmap of active lanes of the GT transceiver quad. Active high.
7:4	`rx_src_rdy_n`	R	0x0	Receive source-ready flag per lane. Active low.
11:8	`tx_src_rdy_n`	R	0x0	Transmit source-ready flag per lane. Active low.
15:12	`tx_dst_rdy_n`	R	0x0	Transmit destination-ready flag per lane. Active low.
19:16	`empty_rxfifo`	R	0x0	Receive FIFO empty flag per lane. Active high.
20	`rd_req_rxfifo`	R	0x0	Read request to the receive FIFO. Active high.
31	`alignment`	R	0x0	Transceiver alignment error. Active high.

GDL Receive Link Status

Table C.6 lists the fields of `LINK_GDL_RX`.

Table C.6: Field descriptions of `LINK_GDL_RX` at offset `0x044`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
3:0	`lane_up`	R	0x0	Bitmap of active lanes of the GT transceiver quad. Active high.
7:4	`rx_src_rdy_n`	R	0x0	Receive source-ready flag per lane. Active low.
11:8	`tx_src_rdy_n`	R	0x0	Transmit source-ready flag per lane. Active low.
15:12	`tx_dst_rdy_n`	R	0x0	Transmit destination-ready flag per lane. Active low.
19:16	`empty_rxfifo`	R	0x0	Receive FIFO empty flag per lane. Active high.
20	`rd_req_rxfifo`	R	0x0	Read request to the receive FIFO. Active high.

GDL Transmit Link Status

Table C.7 lists the fields of `LINK_GDL_TX`.

Table C.7: Field descriptions of `LINK_GDL_TX` at offset `0x048`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
3:0	`lane_up`	R	0x0	Bitmap of active lanes of the GT transceiver quad. Active high.
7:4	`rx_src_rdy_n`	R	0x0	Receive source-ready flag per lane. Active low.
11:8	`tx_src_rdy_n`	R	0x0	Transmit source-ready flag per lane. Active low.
15:12	`tx_dst_rdy_n`	R	0x0	Transmit destination-ready flag per lane. Active low.
20	`wr_req_txfifo`	R	0x0	Write request to the transmit FIFO. Active high.

GRL Receive Link Status

Table C.8 lists the fields of `LINK_GRL_RX`.

Table C.8: Field descriptions of `LINK_GRL_RX` at offset `0x04C`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
3:0	`lane_up`	R	0x0	Bitmap of active lanes of the GT transceiver quad. Active high.
7:4	`rx_src_rdy_n`	R	0x0	Receive source-ready flag per lane. Active low.
11:8	`tx_src_rdy_n`	R	0x0	Transmit source-ready flag per lane. Active low.
15:12	`tx_dst_rdy_n`	R	0x0	Transmit destination-ready flag per lane. Active low.
19:16	`empty_rxfifo`	R	0x0	Receive FIFO empty flag per lane. Active high.
20	`rd_req_rxfifo`	R	0x0	Read request to the receive FIFO. Active high.

GRL Transmit Link status

Table C.9 lists the fields of `LINK_GRL_TX`.

Table C.9: Field descriptions of `LINK_GRL_TX` at offset `0x050`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
3:0	`lane_up`	R	0x0	Bitmap of active lanes of the GT transceiver quad. Active high.
7:4	`rx_src_rdy_n`	R	0x0	Receive source-ready flag per lane. Active low.
11:8	`tx_src_rdy_n`	R	0x0	Transmit source-ready flag per lane. Active low.
15:12	`tx_dst_rdy_n`	R	0x0	Transmit destination-ready flag per lane. Active low.
20	`wr_req_txfifo`	R	0x0	Write request to the transmit FIFO. Active high.

Condensation Point Selection Beta Threshold

Table C.10 lists the fields of `CPS_BETA_THRESHOLD`.

Table C.10: Field descriptions of `CPS_BETA_THRESHOLD` at offset `0x004`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
15:0	`value`	R/W	0x0	Beta threshold for the condensation point selection algorithm.

Condensation Point Selection Isolation Threshold

Table C.11 lists the fields of `CPS_ISOLATION_THRESHOLD`.

Table C.11: Field descriptions of `CPS_ISOLATION_THRESHOLD` at offset `0x008`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
19:0	`value`	R/W	0x0	Isolation threshold for the condensation point selection algorithm.

CaloClusterNet Signal Threshold

Table C.12 lists the fields of `CCN_SIGNAL_THRESHOLD`.

Table C.12: Field descriptions of `CCN_SIGNAL_THRESHOLD` at offset `0x00C`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
15:0	`value`	R/W	0x0	Signal threshold for the CaloClusterNet algorithm.

Belle2Link Packet Version

Table C.13 lists the fields of `B2L_PACKET_VERSION`. The version string is embedded in the header of every Belle2Link packet generated by the firmware and is used by downstream components to verify format compatibility.

Table C.13: Field descriptions of `B2L_PACKET_VERSION` at offset `0x1FC`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
15:8	`major`	R/W	0x0	Major Belle2Link packet version.
7:0	`minor`	R/W	0x0	Minor Belle2Link packet version.

C.3 Belle2Link Registers

The `ctrl_belle2link` submap controls the Belle2Link interface and the associated DAQ path. Table C.14 gives an overview.

Table C.14: Registers of the `ctrl_belle2link` submap.

Offset	Register	Description
`0x00`	`STATUS`	High, if at least one packet missed.
`0x04`	`MISSES`	Missed-packet counter since last reset.
`0x08`	`APPLY`	Commit pending delay configuration.
`0x0C`	`DATA_DELAY`	Data delay in dataframe cycles.
`0x10`	`TRG_DELAY`	Trigger delay in system clock cycles.
`0x14`	`RESET`	Belle2Link subsystem reset.

Belle2Link Status

Table C.15 summarises the individual fields.

Table C.15: Field descriptions of `STATUS` at offset `0x00`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
0	`idle`	R	0x0	System is awaiting an incoming trigger; the DAQ buffer queue is empty. Active high.
1	`overflow`	R	0x0	Sticky overflow flag of the dispatch queue. Requires a full system reset to clear. Active high.
5:2	`active`	R	0x0	Number of currently active DAQ buffers queued for dispatch on the Belle2Link bus.
16	`b2tt_clk_up`	R	0x0	Belle II Trigger Time Module clock present. Active high.
17	`b2tt_up`	R	0x0	Belle II Trigger Time Module operational. Active high.
18	`b2tt_plllkdet`	R	0x0	Belle II Trigger Time Module PLL lock detected. Active high.
19	`b2l_ready`	R	0x0	Belle2Link interface ready. Active high.

Missed Packets

Table C.16 lists the fields of `MISSES`.

Table C.16: Field descriptions of `MISSES` at offset `0x04`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
31:0	`value`	R	0x0	Number of missed packets since last reset. Saturating.

Trigger Delay

Table C.17 lists the fields of `TRG_DELAY`. The value is latched into the pipeline by a write to `APPLY`.

Table C.17: Field descriptions of `TRG_DELAY` at offset `0x10`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
8:0	`value`	R/W	0x0	Trigger delay in system-clock cycles (127 MHz).

Data Delay

Table C.18 lists the fields of `DATA_DELAY`. The value is latched into the pipeline by a write to `APPLY` and is automatically aligned to the beginning of a dataframe by hardware.

Table C.18: Field descriptions of `DATA_DELAY` at offset `0x0C`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
4:0	`value`	R/W	0x0	Data delay in dataframe cycles (8 MHz).

Apply

Table C.19 lists the fields of APPLY. Writing a one latches the current delay configuration into the pipeline; the bit is self-clearing once the values have been applied.

Table C.19: Field descriptions of APPLY at offset 0x08. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
0	value	R/W	0x0	Latches the current delay configuration into the pipeline. Self-clearing.

Belle2Link Reset

Table C.20 lists the fields of RESET. This register resets the Belle2Link subsystem only and does not affect the trigger pipeline.

Table C.20: Field descriptions of RESET at offset 0x14. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
0	value	R/W	0x0	Resets the Belle2Link subsystem. Does not affect the trigger pipeline.

C.4 Trigger Registers

The ctrl_trigger submap provides runtime configuration of the trigger bit forwarded to the GDL via the LEMO connector, as well as a set of saturating 32 bit counters monitoring the individual trigger lines. All counters saturate at 0xFFFF_FFFF and are cleared by a write to TRIGGER_MONITOR_CLEAR. All registers are listed in table C.21.

Table C.21: Registers of the ctrl_trigger submap.

Offset	Register	Description
0x00	TRIGGER_LINE	Select LEMO trigger bit.
0x04	TRIGGER_MONITOR_CLEAR	Clear all trigger monitoring counters.
0x08	TRIGGER_CPT_C2_100MEV	Counter for eclgnn_cpt_c2_100mev.
0x0C	TRIGGER_CPT_C3_100MEV	Counter for eclgnn_cpt_c3_100mev.
0x10	TRIGGER_CPT_C4_100MEV	Counter for eclgnn_cpt_c4_100mev.
0x14	TRIGGER_CPT_C2_125MEV	Counter for eclgnn_cpt_c2_125mev.
0x18	TRIGGER_CPT_C3_125MEV	Counter for eclgnn_cpt_c3_125mev.
0x1C	TRIGGER_CPT_C4_125MEV	Counter for eclgnn_cpt_c4_125mev.
0x20	TRIGGER_CPT_C2_150MEV	Counter for eclgnn_cpt_c2_150mev.
0x24	TRIGGER_CPT_C3_150MEV	Counter for eclgnn_cpt_c3_150mev.
0x28	TRIGGER_CPT_C4_150MEV	Counter for eclgnn_cpt_c4_150mev.
0x2C	TRIGGER_SIG_C2_100MEV	Counter for eclgnn_sig_c2_100mev.
0x30	TRIGGER_SIG_C3_100MEV	Counter for eclgnn_sig_c3_100mev.
0x34	TRIGGER_SIG_C4_100MEV	Counter for eclgnn_sig_c4_100mev.
0x38	TRIGGER_SIG_C2_125MEV	Counter for eclgnn_sig_c2_125mev.
0x3C	TRIGGER_SIG_C3_125MEV	Counter for eclgnn_sig_c3_125mev.
0x40	TRIGGER_SIG_C4_125MEV	Counter for eclgnn_sig_c4_125mev.
0x44	TRIGGER_SIG_C2_150MEV	Counter for eclgnn_sig_c2_150mev.
0x48	TRIGGER_SIG_C3_150MEV	Counter for eclgnn_sig_c3_150mev.
0x4C	TRIGGER_SIG_C4_150MEV	Counter for eclgnn_sig_c4_150mev.
0x50	TRIGGER_CPT_ESUM_1000MEV	Counter for eclgnn_cpt_esum_1000mev.
0x54	TRIGGER_CPT_ESUM_1500MEV	Counter for eclgnn_cpt_esum_1500mev.
0x58	TRIGGER_SIG_ESUM_1000MEV	Counter for eclgnn_sig_esum_1000mev.
0x5C	TRIGGER_SIG_ESUM_1500MEV	Counter for eclgnn_sig_esum_1500mev.
0x60	TRIGGER_LML1	Counter for eclgnn_lml1.
0x64	TRIGGER_LML6	Counter for eclgnn_lml6.
0x68	TRIGGER_LML7	Counter for eclgnn_lml7.

Trigger line selection

Table C.22 lists the fields of `TRIGGER_LINE`. See table C.24 for valid selector values.

Table C.22: Field descriptions of `TRIGGER_LINE` at offset `0x00`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
7:0	`value`	R/W	0x0	Encodes the trigger line routed to the LEMO output.

Trigger monitor clear

Table C.23 lists the fields of `TRIGGER_MONITOR_CLEAR`.

Table C.23: Field descriptions of `TRIGGER_MONITOR_CLEAR` at offset `0x04`. Remaining fields are reserved for future use.

Bits	Field	Type	Reset	Description
0	`value`	R/W	0x0	Clears all trigger monitoring counters. Self-clearing.

Table C.24: Selector values for TRIGGER_LINE.

Selector	Trigger bit
0	tie LEMO output to low
1	eclgnn_cpt_c2_100mev
2	eclgnn_cpt_c3_100mev
3	eclgnn_cpt_c4_100mev
4	eclgnn_cpt_c2_125mev
5	eclgnn_cpt_c3_125mev
6	eclgnn_cpt_c4_125mev
7	eclgnn_cpt_c2_150mev
8	eclgnn_cpt_c3_150mev
9	eclgnn_cpt_c4_150mev
10	eclgnn_sig_c2_100mev
11	eclgnn_sig_c3_100mev
12	eclgnn_sig_c4_100mev
13	eclgnn_sig_c2_125mev
14	eclgnn_sig_c3_125mev
15	eclgnn_sig_c4_125mev
16	eclgnn_sig_c2_150mev
17	eclgnn_sig_c3_150mev
18	eclgnn_sig_c4_150mev
19	eclgnn_cpt_esum_1000mev
20	eclgnn_cpt_esum_1500mev
21	eclgnn_sig_esum_1000mev
22	eclgnn_sig_esum_1500mev
23	eclgnn_lml1
24	eclgnn_lml6
25	eclgnn_lml7
63	tie LEMO output to high

This page intentionally left blank