The FCC software for PED studies

Appendix to the FCC Feasibility Study Report

March, 2025

Principal editors
Brieuc François[1], Gerardo Ganis[1]

Chapter editors
Juan Miguel Carceller[1], Jan Eysermans[2], Thomas Madlener[3], André Sailer[1], Michele Selvaggi[1], Juraj Smiesko[1], Alvaro Tolosa-Delgado[1]

Contributors
Mahmoud Al-Thakeel[1,4], Andrea Ciarma[4], Wonyong Chung[5], Andrea De Vita[1,4], Archil Durglishvili[6], Dolores Garcia[1], Armin Ilg[7], Sungwon Kim[8], Sanghyun Ko[9], Andreas Loeschcke Centeno[10], Giovanni Marchiori[11], Michaela Mlynarikova[1], Lorenzo Pezzotti[4], Louis Portales[12], Edoardo Proserpio[13], Braulio Rivas[14], Jennifer Roloff[15], Romualdo Santoro[13], Swathi Sasikumar[1], Birgit Stapf[1], Zhibo Wu[16], Anna Zaborowska[1]

[1] CERN
[2] Massachusetts Institute of Technology
[3] DESY
[4] INFN
[5] Princeton University
[6] Ivane Javakhishvili Tbilisi State University
[7] University of Zurich
[8] Yonsei University
[9] Seoul National University
[10] University of Sussex
[11] APC, CNRS/IN2P3, Université Paris Cité
[12] CEA Paris-Saclay, IRFU, DPhP
[13] University of Insubria
[14] Escuela Superior Politécnica del Litoral
[15] Brown University
[16] LAPP, CNRS

Contents

1 Introduction

A robust software ecosystem and an adequate level of computing resources are instrumental to fully understand the physics potential of the Future Circular Collider (FCC). Such a Software and Computing system must support all aspects of the research, including event generation, detector simulation, data analysis, and visualisation of simulated data. This comprehensive approach is necessary to explore and maximise the physics reach of the proposed detector solutions.

The workflows to be supported in this phase of the project are illustrated in Fig. 1. The goal of the ongoing studies is to evaluate the response of given detector solutions to the signals and backgrounds that we expect during operations. Backgrounds are mostly of two kinds: luminosity related and beam related. The first ones are those due to the interaction of the colliding particles, and therefore proportional to the luminosity; examples are the two-photon events. The second type of backgrounds is connected to the characteristics of the circulating beams, such as beam losses and synchrotron radiation. To achieve the above we need generators (included those related to the accelerator), detector description, interfaces with codes providing simulation (at various accuracy levels), reconstruction algorithms, analysis tools, also including visualisation. All this is included in the grey box of Fig. 1, which presents what is ran on a given computing unit. In order for this to be possible, we need to provide the code, i.e. build, test and deploy it and to provide resources, in terms of computing units and storage, to process and store the data. These services are shown in the dark grey boxes in the figure.

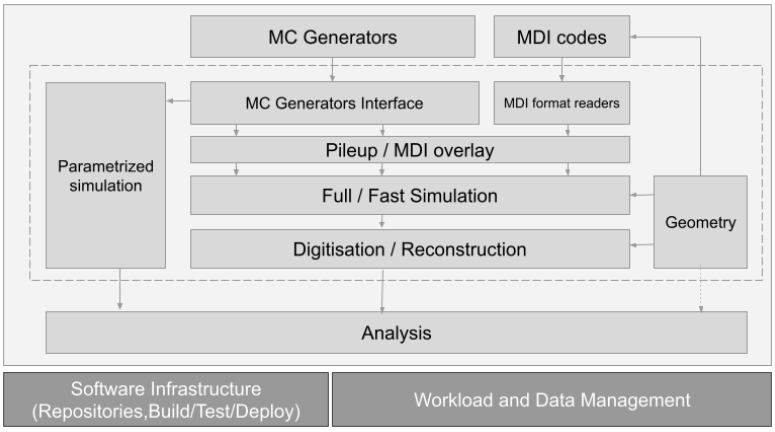


Fig. 1: Workflows to be supported during the project design and FCC Feasibility Study.

The results submitted to the 2020 Update of the European Strategy for Particle Physics, summarised in the FCC CDR, were obtained using FCCSW, a software ecosystem built on the experience of the LHC and integrated with community R&D

projects like DD4hep [1]. FCCSW effectively served the initial needs of the FCC community, particularly for the FCC-hh (the hadron collider) studies, which were the most advanced among those included in the FCC CDR. However, for FCC-ee (the electron-positron collider), most of the studies were conducted using fast simulation or outside of FCCSW.

One of the key priorities in the post-CDR phase was to complete the software ecosystem with all the essential components needed to meet the challenges of FCC-ee. This included proper handling of the complex interaction region and associated backgrounds, a comprehensive suite of e+e- event generators, and an infrastructure that facilitates the implementation of detector concepts, their simulation, and the development of related reconstruction algorithms. Additionally, tools for visualisation, final analysis, and resource management were required. These tools were considered crucial for accurately assessing the physics potential of the proposed experimental infrastructure.

The ambitious plan required human resources far beyond those available during the design phase of the project. As a result, the FCC community eagerly embraced a proposal from other future experiment communities – such as CEPC [2], CLIC [3], ILC [4], SCT [5], and STC [5] – to explore the possibility of developing a common software solution that could be adapted to the specific needs of each community.

The Key4hep [6] initiative, launched during two workshops held between 2019 and early 2020 [7, 8], was built on the successful experience of CLIC and ILC sharing the same software ecosystem (iLCSoft), as well as the common core framework (Gaudi [9]) used by running experiments like LHCb, ATLAS, and BELLE, and developing projects like FCC. The initiative was driven by the belief that the high-energy physics community was ready to further expand the number and role of shared components between experiments.

Software development is inherently a collaborative effort, and Key4hep exemplifies this by maximizing the reuse of established solutions and packages, allowing experiments to benefit from existing community developments. Notable examples include ROOT [10], Geant4 [11], PODIO [12], as well as the previously mentioned DD4hep and Gaudi. By leveraging and contributing to these experiment-independent software packages, Key4hep fosters a broader ecosystem of HEP software, enhancing collaboration and innovation across the field.

The primary aim of this document is to supplement the information presented in the FCC Feasibility Study Report by providing a more detailed account of the status of FCC Software and Computing at the time of the study. It focuses on aspects that could not be extensively discussed in the FCC Feasibility Study Report (FSR), offering a deeper insight into key developments and challenges.

In the remainder of this paper, we explore various aspects of the FCC software environment. Section 2 provides an overview of the Key4hep turnkey software stack, detailing how it supports the FCC software ecosystem by offloading core functionalities to Key4hep while enabling a sharper focus on FCC-specific requirements. The event data model, a critical component integrating various system parts, is discussed in Section 2.1, along with its current status in relation to the needs of the FCC community.

Section 3 describes Delphes, the tool used for parametrised simulation. Sections 4 and 5 cover aspects of geometry implementation and detailed particle simulation. The beam background overlay, digitisation, and reconstruction processes are discussed in Section 6.

The tools available for data analysis, visualisation, and interpretation are introduced in Sections 7 and 8. Finally, Section 9 presents the tools designed to optimise the utilisation of the projected computing resources available for the project and Section 10 summarises the future work.

2 Key4hep

Key4hep is a software framework for future collider experiments. Key4hep integrates all the steps in a typical offline pipeline: generation, simulation, reconstruction and analysis. The components of Key4hep use a common event data model, called EDM4hep, generated from a specification in a text file using Podio. For event processing, Key4hep leverages Gaudi, a proven framework already in use by several experiments at the LHC.

Key4hep is a collaborative effort that benefits communities of several future experiments: FCC, CEPC, CLIC, EIC, ILC or the Muon Collider. Key4hep maximises the reuse of established software solutions in the HEP community, such as ROOT, Geant4, DD4hep or Gaudi. Contributions to Key4hep are mostly experiment agnostic, benefitting a very wide community at once.

2.1 EDM4hep and PODIO

For the modularisation of the different steps a common event data model (EDM) that allows one to exchange data in a well defined way is necessary. For Key4hep this EDM is EDM4hep [13, 14]. The major goal of EDM4hep is to be usable by all communities contributing to the Key4hep effort. It is largely based on LCIO [15], the EDM that has been developed by the linear collider community. Hence, a lot of the experience gained there can be leveraged.

Given its central role in the event processing, EDM4hep needs to be easy to use but should also be performant. This is achieved by using the PODIO EDM toolkit [12, 16, 17]. PODIO allows one to generate a performant and thread safe implementation in C++ from a high level description of EDM4hep in the YAML format. The PODIO package also provides the necessary generic reading and writing functionality that can be used to read any EDM that has been generated via PODIO. Currently PODIO supports several I/O back-ends; among others, the default `ROOT::TTree` [10] based one, as well as one for reading and writing the new RNTuple format [18] are available.

During writing of this report, EDM4hep has gone through several changes to finalise version 1.0. After version 1.0, the goal is to have a strong backwards compatibility guarantee utilizing schema evolution, implemented previously in PODIO.

2.2 `k4FWCore`

`k4FWCore` [19] is the interface between EDM4hep and Gaudi that allows algorithms to have collections of EDM4hep objects as inputs and outputs. It defines the infrastructure that will be used by algorithm developers but it does not deal with the concrete details of the algorithms. `k4FWCore` is a core component of Key4hep, as every algorithm that runs with Gaudi will use something from `k4FWCore`. Two types of algorithms are currently in use: handle-based algorithms and the newer functional algorithms.

2.2.1 Gaudi

Gaudi is an event-processing framework in use in LHCb and ATLAS and other experiments. Gaudi provides tools and services needed to run and monitor algorithms. The

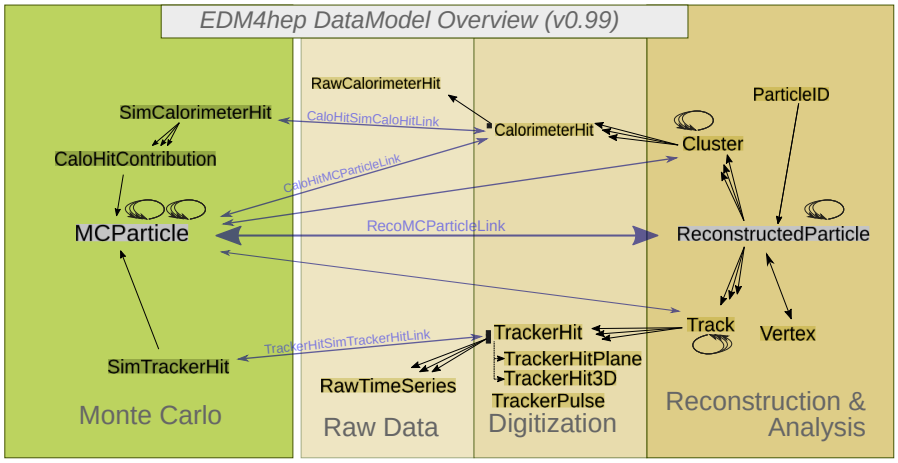


Fig. 2: Schematic diagram of some of the components of the EDM4hep data model for version 0.99.

objects produced by algorithms are saved in one (or multiple in multithreading) Event Stores, that, in a simplistic view, are maps of names to objects. Algorithms use these Event Stores to access data. Gaudi is a comprehensive framework, providing tools for multithreading, auditing and histogramming results.

2.2.2 Handle-based algorithms

The set of algorithms that have been mostly used so far for FCC studies are The algorithms that have been used so far for FCC studies are mostly handle based. In handle-based algorithms, inputs and outputs are obtained through `DataHandle`s. Reading from a file and writing to a file are done by a service called `PodioDataSvc`. Two algorithms are connected to this service: `PodioInput` and `PodioOutput` for reading and writing respectively. The key feature of these algorithms is that a `podio::Frame` is always held by the `PodioDataSvc` that owns all the collections (both read from files and produced by algorithms). When an algorithm runs, it will, through the `DataHandle`s, use the frame held by the `PodioDataSvc` to obtain the inputs. For the outputs, and again using the `DataHandle`s, the ownership of the output collections will be passed to this frame. The `DataHandle`s wrap pointers to collections in a custom wrapper called `DataWrapper` that doesn't own the collections, because otherwise the Event Store from Gaudi would delete them in the clean-up that happens after every event.

For metadata, the `PodioDataSvc` holds a metadata frame and algorithms are able to add or retrieve parameters from it before or after running the main event loop but not during the event loop.

The handle-based algorithms were not designed for running multithreaded. The `PodioDataSvc` doesn't implement the interface needed from Gaudi to be able to use their multithreading tools. To add support for multithreading, a new data service was implemented, to be used together with functional algorithms.

2.2.3 Functional algorithms

Functional algorithms come originally from Gaudi. Gaudi defines a set of base-class algorithms that can be used to define new algorithms by algorithm developers. Functional algorithms do not have internal state. More technically, each algorithm is implemented as a class and the main event loop, `operator()`, is `const`. These algorithms can typically run unchanged in a multithreaded context. Thanks to using functional algorithms, it is easy to use all the functionality in Gaudi provided for running multithreaded, something that wasn't possible with the handle-based algorithms.

Due to the flexibility required for Key4hep, additional features were necessary when implementing support for functional algorithms in `k4FWCore` that can not be found in the original functional algorithms in Gaudi. Some algorithms may produce or take as input an arbitrary number of collections, for example, depending on the detector that they run on. For this reason, two types of functional algorithms (transformers and consumers) were reimplemented in `k4FWCore` including this additional feature. These algorithms are the ones that either create new collections or read existing ones, which cover most of the cases. For the remaining cases that can not be expressed as a transformer or a consumer the plan is to use existing algorithms from Gaudi, without reimplementing existing functionality.

For adding support for reading and writing from files using functional algorithms a new service has been created: `IOSvc`. This service and its two closely related algorithms (a Reader and a Writer), are responsible for reading from files and writing to files. The Reader is in charge of pushing the collections read by the IOSvc and the Writer is responsible for storing collections in a Frame that will later be written to a file. A metadata service has also been developed so that it is possible to add and retrieve metadata parameters as it was done in handle-based algorithms.

Since most of the existing chains of algorithms are written using handle-based algorithms, one important question is whether functional algorithms are compatible with the handle-based ones or not. Thanks to the set of tests in `k4FWCore`, it has been proven that when using the new IOSvc it is possible to combine both handle-based and functional algorithms while reading and writing collections of EDM4hep objects to files.

`k4FWCore` has an extensive set of tests that make use of functional algorithms. The different algorithm types are tested, both in memory and reading and writing to files. There are also tests that take as input or write an arbitrary number of collections, tests that run multithreaded and tests that combine handle-based and functional algorithms in the same chain. In addition, there are tests that check that the script that users need for running algorithms, `k4run`, behaves as specified.

2.2.4 `k4FWCore` components

In addition to the mandatory services to have IO and how to define algorithms, `k4FWCore` also includes a set of optional services that can be used by any algorithm:

- `UniqueIDGenSvc` is a service that produces a seed to be used in a random number generator to guarantee reproducibility. This is done by using some numbers intrinsic

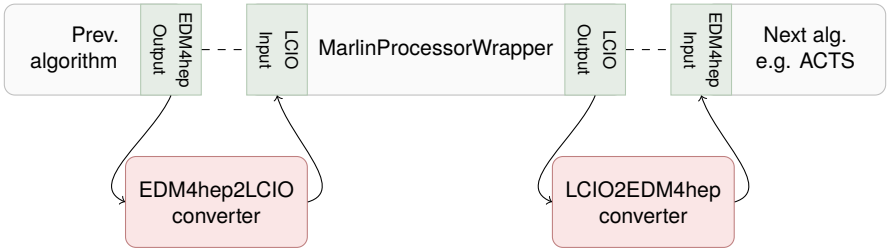


Fig. 3: Schematic diagram of how data gets converted between EDM4hep and LCIO to run Gaudi algorithms and Marlin processors with `k4MarlinWrapper`

to each event and other quantities, such as a combination of the run number, event number and algorithm name. These quantities are combined and hashed together, so that the random number generator will produce the same values for same event when running through the same algorithm independently of, for example, the ordering with respect to other events.

- `EventHeaderCreator` is an algorithm that creates event headers and fills them with a run number and an event number.
- `CollectionMerger` is an algorithm that combines several EDM4hep collections into a single one.
- Interfaces: a common set of interfaces that algorithm developers can implement downstream and use in several repositories.

2.2.5 Usage

`k4FWCore` provides a script for users to run algorithms: `k4run`. When given a *steering file*, `k4run` will show the different options that have been provided (or their default values) and will call Gaudi to run algorithms as specified in the steering file. In addition, `k4run` allows users to change parameters passed to algorithms and services from the command line, without having to edit any file.

2.3 The Marlin wrapper

One of the key components of Key4hep is the Marlin wrapper, available in the `k4MarlinWrapper` repository. This wrapper integrates Marlin processors into Gaudi algorithms, enabling them to run within the Gaudi framework. Internally, events are converted between `EDM4hep` and `LCIO`, the event data model used by the Linear Collider community. These conversions occur as needed, depending on whether the next processing step uses a Gaudi algorithm or a Marlin processor (see Fig. 3). However, this introduces both time overhead due to conversions and additional memory usage since events are duplicated during the process.

2.4 The Key4hep stack

The Key4hep stack is the set of all the packages needed to work in Key4hep. It is a collection of around 600 packages built with Spack [20] that are deployed to CVMFS [21],

although most of the packages are dependencies of the packages that most users work with. Setup is easily done with a single command, and several operating systems are supported. As a core component of Key4hep, the Key4hep stack lowers the threshold needed to get started since, depending on the task, users may not have to compile anything themselves.

2.4.1 Building with Spack

Spack is the tool used to build the Key4hep stack. Spack is both a repository with thousands of recipes for almost every software package available and code to build these recipes. The library of recipes is public and anyone can contribute to it, which has proven to be a time-saving approach. Other people can and will fix recipes that are in use in Key4hep while the time we have invested in fixing recipes also helps other people. Our complete Spack setup also includes another repository called `key4hep-spack` [22] where recipes for packages that are not yet in the main Spack library can be found as well as the configuration needed for Spack to build the Key4hep stack.

Two types of builds are done for Key4hep. The first one is stable releases, which can be found under `/cvmfs/sw.hsf.org` . These releases are done every few months on demand when there are major changes or they are needed for any particular reason, for example, to preserve the status of the software used to produce a certain data set. The stable releases only use tagged versions of the packages and are, most of the time, a complete build from scratch of all the packages. Nightly builds are available under `/cvmfs/sw-nightlies.hsf.org` and are built in two steps. First, a complete build from scratch of all the packages is done, using the latest commit of a subset of the packages while for the rest the latest tagged version is used. Then, for the following weeks, every day a new build is made on top of this build from scratch, which will only rebuild whatever packages have changed and their dependents. This is an acceptable compromise between build time, maintainer time and stability, because Key4hep packages (and other packages) will be updated every day, but other big packages like ROOT or Geant4 will be updated only when a new build from scratch is done. Unlike stable releases, nightly builds are deleted from time to time and are not intended for use in production.

Additional files are deployed with each build to CVMFS, which are needed to reproduce a build. There are three: a file with the commit of Spack that was used to make the build, another one with the commit of `key4hep-spack` that was used, and one with the cherry picks that have been incorporated into the current version of Spack. This is typically the case when making a new build from scratch; fixes are needed and pull requests with the fixes are made to Spack so they need to be cherry-picked into the current build to make it work. With these three files it is possible to recover the exact state of the Spack repository and `key4hep-spack` that were used to make the builds, in case they had to be reproduced in the future.

Several operating systems are supported: AlmaLinux 9 and Ubuntu 22.04, and until June 2024, also CentOS 7. The stable releases are done in the CMake `Release` mode, that is, optimised and without debug symbols, while the nightlies are done in the `RelWithDebInfo` mode with both optimisations and debug symbols. In the future builds will be done one time in `Release` mode and another in `Debug` (no optimisations

but with debug symbols) mode, so that debug builds can be used when developing and debugging while `Release` builds can be used when better performance is needed. At the time of writing, GCC 14 has been added to AlmaLinux 9 in the nightly builds although builds with the system compiler, GCC 11, are still available. For Ubuntu 22.04 the builds are done using the compiler provided by the OS (GCC 11).

2.4.2 Testing, validation and continuous integration

After a build has been completed, a set of tests is run. These tests come mostly from issues that users have had in the past, and the build is only deployed if every package built successfully and all of these tests pass. In addition, in many of the Key4hep repositories, there is a workflow that runs the tests of each repository using the current nightly build.

Validation of the reconstruction chain is also performed. Every night a sample is simulated and reconstructed to validate the full simulation and reconstruction chain. Analysis scripts run after, creating plots that are deployed in a webpage where the current samples and a reference sample are compared. This validation is running every night but with a limited set of detectors, currently only CLIC, although CLD, IDEA and ALLEGRO will be added soon. This validation is applied to as this can be an useful tool for detector studies, validating the status over its development and changes over time.

Fast workflows were developed for Continuous Integration (CI). Every time there is a pull request on the Key4hep repositories, they will be built both on top of the latest stable release and the latest nightly. These workflows use a build cache that was produced by an earlier run of the tests, so the build process is very quick, which means that most of the time spent by the CI is typically the duration of the tests. In addition, there is another workflow to test how the current changes in a pull request affect dependent repositories. This workflow also features ways of setting up a coherent build with several pull requests in different repositories by parsing the text of the pull request. In practice this feature is complex to use, and people who work on changes that affect many repositories typically make these builds themselves so it is rarely used.

Several of these workflows are centralised: they live in a common repository, `key4hep-actions` [23], and, using custom tools, they are pushed to all the relevant Key4hep repositories, guaranteeing consistency and easing maintainability since they only need to be changed in one place. There are plans to centralise more workflows, like code formatting.

2.4.3 Usage and user support

The interaction between the Key4hep stack and the user starts at setting up the Key4hep stack. With two simple commands, the user can source a script to set up the stable releases or the nightlies that will automatically detect which operating system they are running on and, based on that, set up the Key4hep stack accordingly:

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

or

```
source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
```

After sourcing, a message appears with important information, such as where to ask for help and what to do to reproduce the current environment in the future. The default of the scripts is to source the latest build, so, for example, for the nightly builds the build done the current day will be used. Several command line parameters can be passed to the setup scripts: `-r` with the name of a release (typically a date) to set up that release, `-c` to select the compiler, and other parameters to list the existing releases and packages.

Some help is provided for developing when using the Key4hep stack. The setup scripts define a function called `k4_local_repo` that, when called from a repository, will clear the environment from all the paths with the same name as the current repository and will add paths to point to the local installation. For most packages this is enough to set up the environment so that the current package is used instead of the one that could be found in the Key4hep stack from CVMFS.

When users have issues, they typically report them in the `key4hep-spack` repository. There, there is also a pinned issue with the known problems so that users can know what is not expected to work. There is also a mailing list for any Key4hep-related topics.

2.5 DD4hep philosophy and internals

Consistently, efficiently, and conveniently describing the geometry of the detectors being considered for the FCC is of utmost importance for successful studies. DD4hep [1, 24–26] was created to enable the consistent and convenient description of detectors. It is designed around providing a single source of information of geometry and related information for simulation, reconstruction, and analysis. DD4hep can provide a complete detector description including the geometry, materials, visualisation, readout, sensitive detector configurations, alignment and calibration. Through these features, DD4hep supports the full experiment life cycle from detector concept development, detector optimisation, to construction and operation, and offers a seamless transition from one stage to the next.

DD4hep itself consists of multiple packages: DDCore provides geometry handling, DDG4 [27] offers the interface to Geant4 and includes the `ddsim` interface for configuring and running Geant4 simulations, DDCond and DDAlign offer interfaces to access condition and alignment information [28], and DDRec [29] offers a more abstract interface and convenience functions that reconstruction algorithms can take advantage of.

The geometry description in DD4hep is delegated to ROOT's geometry system – and to Geant4 [11, 30, 31] when running simulation. The geometry is created out of logical volumes that are placed inside or next to one another to create complex geometries. In general a complete detector consists of multiple sub-detectors, where each sub-detector is created by a specific detector constructor implemented in C++ and a *compact* XML file that contains various parameters that the detector constructor interprets. The combination of C++ drivers and parameters in XML allows for the creation of generic and re-usable detectors, where the modifications of a few numbers or characters in an XML file can change the dimensions, number of layers, or materials

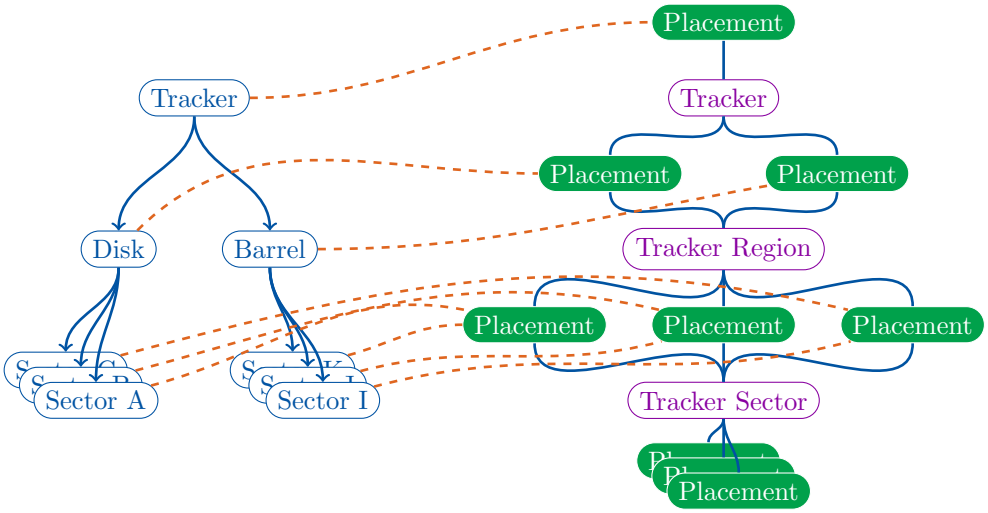


Fig. 4: Relationship between the detector element tree on the left and the geometry hierarchy on the right hand side. Each blue box represents a Detector Element belonging to a specific placement.

of a sub-detector. The sub-detector geometries implemented for the Feasibility Study are described in Section 4.

In addition to the geometry, the detector description in DD4hep allows one to attach *detector elements* to different pieces of a detector. Usually these pieces are sub-detectors and parts of sub-detectors. A detector element points to placed logical volumes from the geometry tree and contains environmental conditions and other properties such as alignment or readout information. In particular DD4hep allows one to attach arbitrary *extensions* – in the form of C++ classes – to any detector element at any time. While the detector geometry allows one to place volumes repeatedly, the detector element tree is a directed acyclic graph. Figure 4 shows the relationship between the detector element tree and the geometry graph.

2.5.1 DDG4: the gateway to Geant4

As mentioned above, DD4hep includes the transformation of the TGeo based core geometry to a Geant4 based geometry through the DDG4 package. This in-memory translation converts also the materials, limits and regions, and instantiates magnetic fields. The configuration of Geant4 simulation is handled through DD4hep's plug-in mechanism. All plug-in properties can be configured through XML, Python or ROOT-AClick. The plug-ins allow one to configure the Generation, Run Actions, Event Actions, Tracking Actions, Sensitive Detectors and Physics Construction. Out of the box, DDG4 includes many plug-ins, so that in most cases, no additional user plug-in are necessary to start simulations with Geant4. For primary particles, DDG4 handles the treatment of long lived particles and provides access to particle gun and general particle sources. It can read the Monte Carlo records from HepMC, stdhep, HepEvt

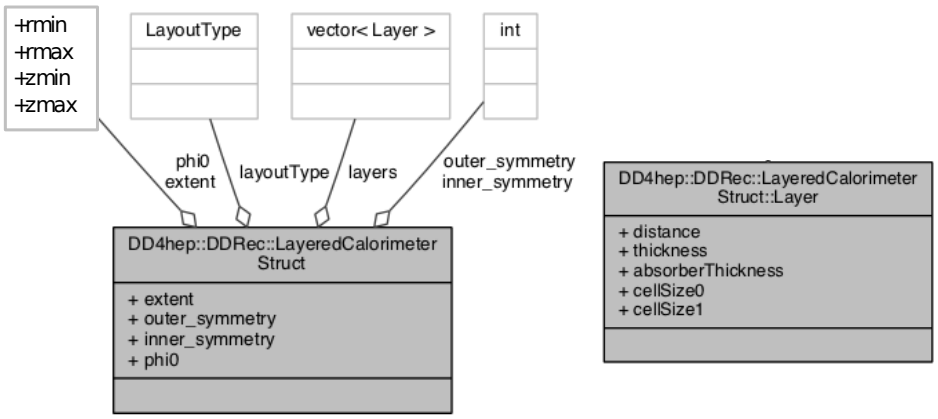


Fig. 5: Unified Modeling Language diagram for the `DDRec::LayeredCalorimeterData` struct in DD4hep.

and SLCIO files. Sensitive detectors are available for silicon or gaseous detector tracking detectors and for calorimeters. A configurable MCTruth History handling allows one to perform various studies through the association of hits and Monte Carlo particles. Physics lists and extensions can be configured, for example for Optical Photons necessary to study scintillator or Cerenkov based detectors. More information about configuring full simulation with DD4hep is given in Section 5.

2.5.2 DDRec: high level reconstruction information

While the geometry used for simulation can be extremely detailed, the reconstruction algorithms usually require a more high level view of the geometry. For this purpose DDRec contains abstracted data structures that offer an interface for different geometries to represent themselves uniformly to the reconstruction algorithms. These data structures are attached to the detector elements either at the construction of the geometry or dynamically through plug-ins. As an example, Fig. 5 shows the Unified Modeling Language diagram of one the DDRec data structures for layered calorimeters. It contains basic information such as the inner and outer radii, positions along Z, the number of layers. For each layer the distance to the interaction point, the thickness of the layer and the thickness of the absorber, as well as the cell sizes are stored.

Apart from specific data structures for tracking detectors, DDRec also contains a *Surface* class which represents individual sensor modules and offers interfaces necessary for track reconstruction such as global to local co-ordinate transformations. These surfaces are again attached to detector elements at construction or at runtime through plug-ins, allowing great flexibility to configure surfaces according to the needs of reconstruction algorithms.

2.6 Outlook

For the first time, EDM4hep is very close to a stable version that will have to be thoroughly tested. Schema evolution will be exercised to make sure that backwards compatibility is still maintained after the unavoidable changes that will happen in EDM4hep.

Regarding the framework core, `k4FWCore` and Gaudi, full simulation chains for the main detectors are being developed but are not there yet, so most of the work will concentrate on the actual developing of the algorithms. Once these chains are ready, it will be possible to leverage the multithreading capabilities of Gaudi and their support in `k4FWCore` to make them very performant.

3 Parametrised simulation

Delphes is a framework for multi-purpose detector fast simulation [32]. Instead of simulating the time-consuming interactions of stable particles with matter (as done in Geant4 [31] for example), Delphes makes use of a parametrised detector response in the form of resolution functions and efficiencies. The Delphes simulation includes a track propagation system embedded in a magnetic field, electromagnetic and hadron calorimeters, and a muon identification system. Physics objects that can be used for data analysis are then reconstructed from the simulated detector response. These include tracks and calorimeter deposits and high level objects such as isolated leptons, jets, and missing momentum. Delphes also includes a simplistic particle-flow reconstruction that combines optimally tracks and calorimeter information to form particle candidates. Such particles are then used as input for jets clustering, missing energy, and isolation variables. A detailed description of the Delphes framework can be found in Ref. [32]. An algorithm for reconstructing the track parameters and covariance matrix of charged particles for an arbitrary tracking sub-detector geometries has been recently developed [33]. Additional modules allowing for particle identification using time-of-flight [34] and ionizing energy loss [35] information have been implemented.

The FCC-ee Delphes detector configurations can be found in the official Delphes release [36] as well as in the FCC-config software webpage [37]. The benchmark detector used for physics studies is the IDEA detector with ECAL crystals. An alternative configuration that includes the CLD silicon tracker has also been developed [38] and used for a small fraction of physics analyses.

3.1 Tracking

The first step carried out by Delphes is the propagation of long-lived particles within a uniform axial magnetic field with $B_z = 2$ T. The magnetic field is assumed to be localised in the inner tracker volume. The tracker geometry is defined in the configuration file whereby each layer placement, position resolution and material budget is specified. The geometries and corresponding material budgets are given respectively in Fig. 6 and Fig. 7.

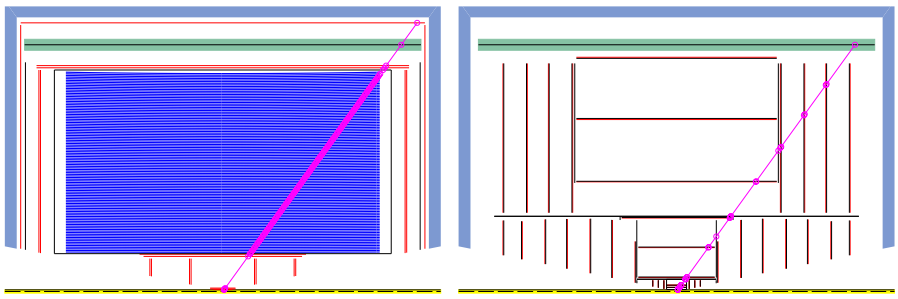


Fig. 6: Geometries of the IDEA (left) and CLD (right) trackers. A 45 degree track is drawn for illustration purposes.

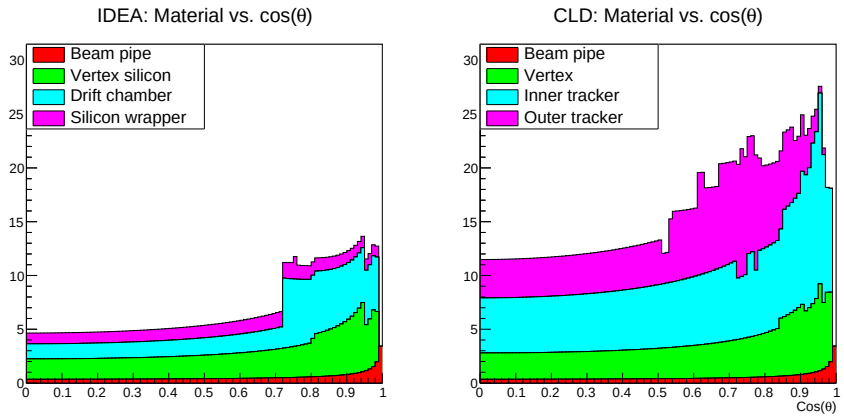


Fig. 7: Material budget in the IDEA (left) vs CLD (right) tracker Delphes implementations.

Charged tracks follow a helix trajectory that is described with a set of five parameters: $\vec{\alpha} = (D, \phi_0, C, z_0, \lambda)$. These parameters are defined at the point of closest approach (PCA) of the track to the z-axis; D is the signed transverse distance of the PCA from the z-axis, ϕ_0 is the track azimuthal angle, C is the signed half curvature, z_0 the z-coordinate of the PCA and λ the cotangent of the track polar angle. The full track covariance matrix as a function of momentum and angle is predicted. The obtained resolutions as a function of momentum of the track parameters $(\mathrm{p}, D, z_0, \theta)$ for the two different reference detector configurations are shown in Fig. 8. A random smearing based on the 5 track parameters is applied to simulate the track fitting. Notably, we stress that the track PCAs and their corresponding covariance matrix terms are crucial inputs to the jet flavour tagging algorithms. Their detailed description is given elsewhere [39, 40].

During its motion the charged particle will cross some of the layers described in the geometry and is reconstructed as a track provided that it produces at least 6 hits [1].

3.2 Calorimetry

After their propagation in the tracker magnetic field, long-lived particles reach the calorimeters. The electromagnetic calorimeter, ECAL, is responsible for measuring the energy of electrons and photons, while the hadron calorimeter, HCAL, measures the energy of charged and neutral hadrons. For the FCC-ee physics in Delphes, the calorimeters have a finite lateral segmentation in pseudo-rapidity and azimuthal angle (η, ϕ) and no longitudinal segmentation. The transverse segmentation of the FCC-ee calorimeters is uniformly distributed in (η, ϕ) and equals to 0.02. At 2 m from the IP, this corresponds to 5x5 cm cells. The coordinate of the resulting calorimeter energy deposit, the tower, is randomly smeared across the cell.

[1] The minimum number of hits for a track to be reconstructed is a configurable parameter in Delphes.

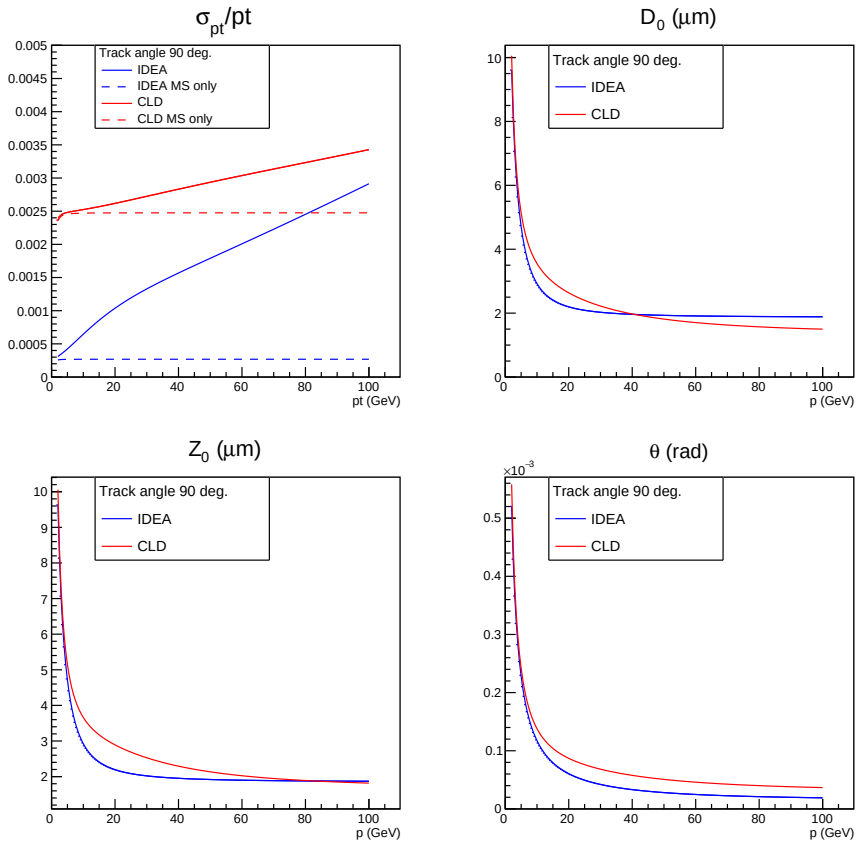


Fig. 8: Track parameters resolution for the IDEA and CLD detector concepts for FCC-ee [41]. The dashed lines in the top left plot show the multiple scattering contribution.

The resolutions of ECAL and HCAL are parametrised as a function of the particle energy:

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{N}{E}\right)^2 + \left(\frac{S}{\sqrt{E}}\right)^2 + C^2\,, \tag{1}$$

where N, S and C are respectively the *noise*, *stochastic* and *constant* terms. The baseline calorimeter is assumed to be made of crystal for the ECAL and Dual Readout for the HCAL.

The electromagnetic and hadronic energy deposits are independently smeared by a log-normal distribution. For simplicity, it is assumed that electromagnetic particles such as electrons, photons and neutral pions (if not decayed by the Monte Carlo event generator) exclusively deposit their energy in the ECAL. Likewise, long-lived hadrons (such as charged pions, kaons, protons) only deposit in HCAL and muons do not leave

any energy deposit in the calorimeters. The values of S and C adopted for the FCC-ee calorimeters are summarised in Table 1. A complete description of the calorimetry requirements of the FCC-ee can be found elsewhere [42].

Table 1: Resolution parameters of the electromagnetic and hadronic calorimeters. N, S and C are respectively the *noise*, *stochastic* and *constant* terms. The assumed functional form for the energy resolution is given in Eq. (1).

	Noise (N)	Stochastic (S)	Constant (C)
ECAL	5 MeV	3%	0.2%
HCAL	10 MeV	30%	5%

3.3 Particle-flow reconstruction

In Delphes, the particle-flow event reconstruction is based on a simplified approach that uses as input the tracks and the calorimeters towers.

If the momentum resolution of the tracking system is better than the energy resolution of calorimeters (typically for momenta below some threshold), it is convenient to measure the charged particles momenta using the tracking information. Vice-versa at high energy, calorimeters provide a better momentum measurement. The particle-flow algorithm exploits this complementarity to provide the best possible single charged particle measurement — the *particle-flow tracks*. These contain electron, muons and charged hadrons. Thanks to the excellent tracking resolution assumed for the FCC-ee detector, the particle-flow charged particle resolution is mainly driven by the tracking. Charged candidates are given pion mass by default, unless they are identified as a muon or an electron.

In Delphes, a lepton (electron or muon) from the *particle-flow tracks* collection, or a photon from the *particle-flow photon* collection has a probability of being identified. For charged particles the identification efficiency is applied on top of the tracking efficiency discussed in the previous section. The assumed identification efficiencies for electron and muons are 99%, provided they have a transverse momentum greater than 2 GeV and 0.65 GeV respectively, i.e. the minimum momentum needed to reach to the Muon and ECAL detectors respectively.

If a significant neutral energy deposit that exceeds the charged deposited energy is detected in a given calorimeter tower, *particle-flow photons* or *particle-flow neutral hadrons* are formed. Together with the *particle-flow tracks*, they form the (generically labelled) *particle-flow candidates* collection and are used as an input for electrons, muons and photons identification and isolation, as well as jets and missing momentum.

In the baseline IDEA configuration particle-flow candidates originating from electrons, photons and muons are identified respectively as electrons, muons and photons with a 99% probability. The fake identification probability is 0%.

The performance of the Delphes particle-flow algorithm is displayed in Fig. 9, where the visible mass distribution in $H \to s\bar{s}$ events obtained with particle-flow is compared to a pure calorimeter based reconstruction. The simplistic simulation and reconstruction paradigm adopted in Delphes does not include processes that require a detailed description of the particle-detector interaction, e.g. photon conversions, nuclear interactions, bremsstrahlung, and beam induced backgrounds. Only a full fledged particle-flow algorithm based on full-simulation will predict the ultimate jet performance under the FCC-ee conditions. The performance of the Delphes particle-flow algorithm should be understood as the ultimate asymptotic target for the FCC-ee performance.

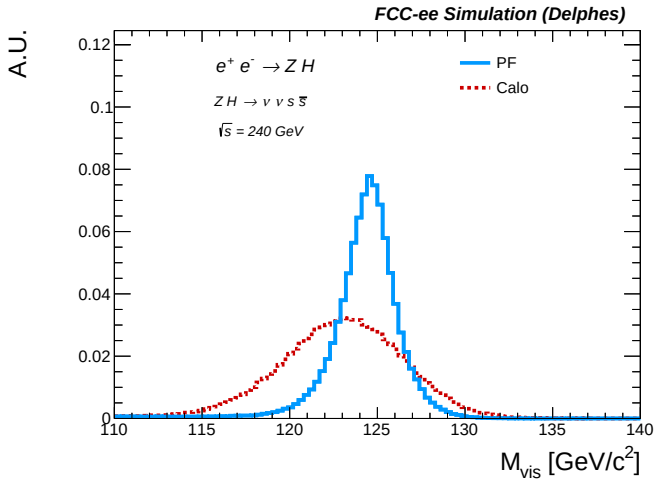


Fig. 9: Reconstructed visible mass for $\nu\nu H(s\bar{s})$ events at $\sqrt{s} = 240$ GeV using particle-flow reconstruction in Delphes and calorimeter standalone reconstruction.

3.4 Particle identification

Two complementary particle identification techniques have been included in the Delphes fast simulation package. The precise measurement of the time of arrival of tracks in the outermost part of the tracking volume, together with the momentum and the path length, provide an indirect measurement of the particle mass via the well known time-of-flight method. This method has been implemented in the `TimeOfFlight` module [34]. The cluster counting method, dN/dx, implemented in the `ClusterCounting` module [35], consists of counting the multiplicity of the primary ionisation clusters produced along the track in gaseous detectors, which together with the particle momentum, can also be used to infer the particle mass. In this section we discuss the implementation of these two methods within the simulation framework.

3.4.1 Time-of-flight

The time-of-flight (t_{flight}) of a particle can be expressed as:

$$t_{\text{flight}} \equiv t_{\text{F}} - t_{\text{V}} = \frac{L}{\beta} = \frac{L\sqrt{p^2 + m^2}}{p} = \frac{LE}{\sqrt{E^2 - m^2}}, \tag{2}$$

where t_{F} is the measured time after propagation, t_{V} is the particle time of production at vertex, L is total path length, and p, E and m are the momentum, energy and mass of the particle, respectively. Provided that the quantities L and p (or E) and t_{V} can be measured, the measurement of t_{flight} provides an estimate for the particle mass and thus a powerful handle for particle identification.

For charged particles the reconstructed mass is given by:

$$m^{(c)}_{\text{t.o.f.}} = p\sqrt{(\frac{t_{\text{flight}}}{L})^2 - 1}, \tag{3}$$

The initial position (and therefore L) and the particle momentum p are reconstructed by means of the inner/outer tracking system, and simulated with the procedure described in Section 3.1. The time of a particle production at vertex t_{V} can be estimated indirectly. In the physics studies we assume we are able to reconstruct the initial time at vertex perfectly and therefore we take the initial time from Monte Carlo simulation.

The time of flight distribution of charged Kaons and Pions emitted at 90 ° is shown in Fig. 10 (Left), assuming a 30 ps timing resolution, which allows for an efficient 3σ K/π discrimination for momenta $p < 3.5$ GeV. For reference, a 3 ps timing resolution leads to 3σ separation for momenta $p < 10$ GeV.

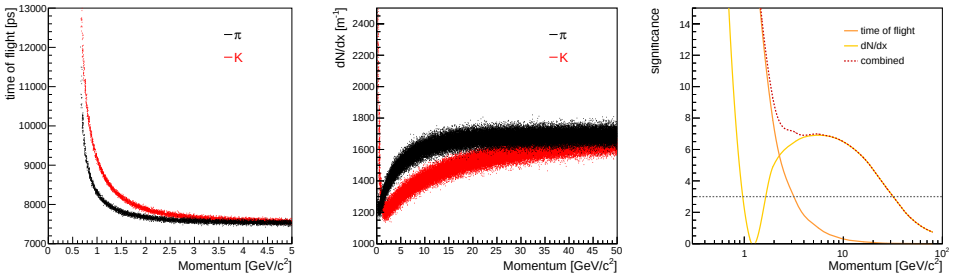


Fig. 10: Left: Time-of-flight for $K^{\pm}$ and $\pi^{\pm}$ track at $\theta = 90\,^{\circ}$ as a function of momentum in the IDEA detector drift chamber. Center: Reconstructed $m_{\text{t.o.f.}}$ for $K^{\pm}$, $\pi^{\pm}$, K_L, protons and neutrons with momenta $p = 1$ GeV. Right: K/π separation in number of σ as a function of the particle momentum using the dN/dx and time-of-flight methods.

3.4.2 Cluster counting

The cluster counting technique is expected to provide improved particle identification relative to the more commonly used dE/dx methods. The number of ionisation clusters per unit length is obtained from Garfield simulation [43]. An array of number of ionisation clusters per unit length for several values of $\beta\gamma$ is obtained from Garfield and used to interpolate the average cluster density. The total mean number of clusters is found by multiplying for the track length in the chamber. Finally the observed cluster number is obtained by extraction over a Poisson distribution with that mean. Four common gas options are available: pure helium, pure argon, 90% helium/10% isobutane or 50% argon/50% ethane. This library can be easily extended if needed to a larger collection of gas mixtures.

In Fig. 10 (centre) the potential for K/π separation is shown for a 90% helium/10% isobutane mixture over a wide range of momenta ($2 < p < 30$ GeV) . The combination of the cluster counting and time-of-flight techniques is displayed in Fig. 10 (right) and shows an efficient separation of $K^{\pm}$ / $\pi^{\pm}$ separation ($\geq 3\sigma$) for momenta $p < 30$ GeV.

3.4.3 Missing momentum, jets and flavour tagging

The missing energy is computed as the magnitude of the total transverse momentum carried by the full list of reconstructed particle-flow candidates. The Delphes framework integrates the FastJet package [44], allowing for jet reconstruction with the most popular jet clustering algorithms. In the FCC-ee detector configuration jet collections using the Durham algorithm [45] in exclusive N mode is mostly used. As mentioned earlier, jet collections for several requested number of jets N are formed using particle-flow candidates.

The ParticleNetIDEA jet flavor tagger [40] is then used to assign each a probability of originating from a given flavour among 7 possible categories: g, u, d, s, c, b, τ. The tagger uses the information of all particle-flow candidates within a jet, including kinematic, displacement and particle identification. The performance and implementation of the algorithm is explained in detail in Ref. [40].

3.5 Delphes integration into Key4hep

In order to integrate more seamlessly with the rest of the Key4hep software stack the k4SimDelphes [46] package has been created. Its main goal is to provide the output of a Delphes parametrised simulation in the EDM4hep format. In the following paragraphs we provide a brief description of the structure of the output as well as an introduction to the basic features.

3.5.1 Basic features and functionality

k4SimDelphes wraps the core parts of Delphes and provides the additional necessary functionality to convert from the Delphes data model to EDM4hep at the end. It reimplements parts of the reading functionality for the various input formats that Delphes also supports. Additionally, it provides some functionality for interfacing Pythia8 [47] and EvtGen [48], which is described below. k4SimDelphes runs with standard Delphes

cards without requiring any changes to them provides standalone executables as well as integration into the Gaudi based framework of Key4hep (see Section 2).

The standalone executables read generator outputs in various formats and directly produce EDM4hep output files. Currently most of the input formats that are also supported by Delphes are also supported by k4SimDelphes, namely the Delphes format itself, StdHep, and HepMC. Additionally, through the Pythia8 interface, it can also read Pythia8 input cards and run the generation on the fly. The supported formats have mainly been driven by the needs of the community so far and adding support for more is possible.

3.5.2 Structure of the EDM4hep output

The core functionality of k4SimDelphes is a converter library that converts the Delphes data model to EDM4hep in memory. The main feature of this converter library is that it creates one collection of EDM4hep `ReconstructedParticle`s that comprises all individual reconstructed particles as is typically possible in lepton colliders. This collection is a combination of several Delphes collection, where the resulting object in EDM4hep format depends on the Delphes type. Particle collections for which Delphes runs a dedicated identification, e.g. Muons, Electrons or Photons, are stored as subset collections that simply point into the collection of all reconstructed particles. Jet collections are also stored as separate collections, where the constituents are again from the global reconstructed particle collection. Additionally, some event level Delphes information is also converted to EDM4hep collections. An overview of the input and output data types for the conversion can be seen in Table 2.

Table 2: Overview of the conversion functionality of k4SimDelphes showing which Delphes data model classes are converted into which EDM4hep types.

Delphes	EDM4hep
`GenParticle`	`MCParticle`
`Track`	`ReconstructedParticle` with associated `Track`
`Tower`	`ReconstructedParticle` with associated `Cluster`
`Jet`	`ReconstructedParticle`
`Muon`	`ReconstructedParticle` (subset collection)
`Electron`	`ReconstructedParticle` (subset collection)
`Photon`	`ReconstructedParticle` (subset collection)
`MissingET`	`ReconstructedParticle`
`ScalarHT`	`ParticleID`
`ParticleFlowCandidate`	`ReconstructedParticle`
n/a	`MCRecoParticleAssociation`

For DelphesTracks and `Tower`s most of the kinematic information is stored directly in the EDM4hep `ReconstructedParticle`s that are created from them. Some additional information is stored into the EDM4hep `Track` or `Cluster` that is attached to them. For tracks this effectively comprises the track parameters as well as the covariance matrix. For clusters the energy is duplicated and additionally a `CalorimeterHit`

is created and attached to the `Cluster` to store the position information available from the Delphes `Tower`.

Since in EDM4hep no direct connection between generator and reconstruction level data exists, the information of which `ReconstructedParticle` originated from which `MCParticle` is stored in a separate collection of `MCRecoParticleAssociation`s that link the two worlds.

In order to allow working with different Delphes configurations which might produce different outputs the conversion library is configurable on the input as well as the EDM4hep output side. The main customisation points are: which collections to convert overall, which collections to collect into the global collection of reconstructed particles and the names of the output collections. A default configuration is provided as part of a k4SimDelphes and has been used for production runs.

3.5.3 EvtGen interface

In order to perform heavy flavour studies k4SimDelphes has also been interfaced to the EvtGen MC generator [48]. Two dedicated readers are available. `DelphesPythia8EvtGen_EDM4hep` simply uses the interface between Pythia8 and EvtGen which allows to force the decay of heavy flavour states that are produced by Pythia8 in a configurable set of decay channels. A second version, `DelphesPythia8EvtGen_EDM4hep_k4Interface`, has been implemented as well to increase the efficiency of generating and simulating events for flavour studies. Compared to the first version it can be used to force a complete re-hadronisation of the event in case none of the desired heavy flavour states was generated by Pythia8 without having to regenerate a new signal event.

3.6 Outlook

Delphes, as well as the k4SimDelphes conversion library have been rather stable the last few years and apart from bug fixes and regular maintenance to keep up with the developments in EDM4hep, no significant developments are currently foreseen for these parts. The main foreseeable work will most likely go in adding support for reading more formats for the standalone executables, e.g. HepMC3 [49]. On the other hand some generic generator interface functionalities would be better suited in a dedicated package and are part of k4SimDelphes mainly for historical reasons. Moving this functionality to a more appropriate place would make it possible to also use them for full simulation studies.

4 Detailed detector geometry descriptions

The parametrised simulation described above provides a highly flexible and light-weight tool to perform various physics studies but some applications require simulations with a higher level of details. For instance, detector R&D activities and physics analyses accessing detector level information both need a detailed description of the detector geometries and of their response. To meet these needs and to validate the detector parametrisation described in the previous section, a complete chain of detailed simulation of the FCC (sub-)detectors is under active development. This section describes the status of the geometry implementation for the various sub-detectors currently under consideration for FCC-ee.

As introduced in Section 2, the DD4hep toolkit [1] is used to model FCC detector geometries. In DD4hep, the detector geometry is described using a *compact* XML file, which provides a global vision of the detector. This compact file is where detector factories are called to build each sub-detector. A dedicated detector factory is employed to load CAD shapes, which is particularly important when detailed support structures are needed, such as in case of the beam pipe (Section 4.1). The material description and its visualisation options, along with other simulation-specific parameters such as the optical properties, regions, user limits, magnetic fields, readout structures and segmentations are also given in the compact file. Since the DD4hep geometry relies on ROOT, DDSim (see Section 5.1) translates it into Geant4 format for simulation.

The various geometries are hosted in the `k4geo` gitHub package [50] which also hosts the detector models of ILC and CLIC. Each (sub-)detector implementation is versioned with the following appendix "oX_vY" where X labels detector variants that will be studied in parallel (all of them being "concurrent" candidates) while the Y number is incremented e.g. when the modelling of the detector is improved and the new version is meant to permanently supersede the previous one.

4.1 Beam pipe

The detailed description of the FCC-ee interaction region (IR) beam pipe can be found in [51]. For the CDR studies, a parametric model of the beam pipe in the detector region was developed based on native shapes available in DD4hep. The current version of this model features the double layered AlBeMet central chamber with the paraffin cooling and the gold coating, and a conical chamber to reproduce the beam pipe separation region, as shown in the left side of Fig. 11. This model can be found under the folder `FCCee/MDI/compact/MDI_o1_v00/` in k4geo [50].

In order to have a more accurate description of the beam pipe, the engineered CAD models can be directly imported in DD4hep. The main differences between the CAD and the shape based model are the shape of the conical chamber and beam pipe separation region, that in the CAD is more ellipsoidal and with a smooth transition following the low impedance design described in [52], and the cooling manifolds for both the central and elliptoconical chamber. The comparison of the two models is shown in Fig. 11.

Implementation of the engineered beam pipe CAD models in DD4hep is performed via the `CAD_multivolume` shape type, which allows the import of `.stl` mesh files. This

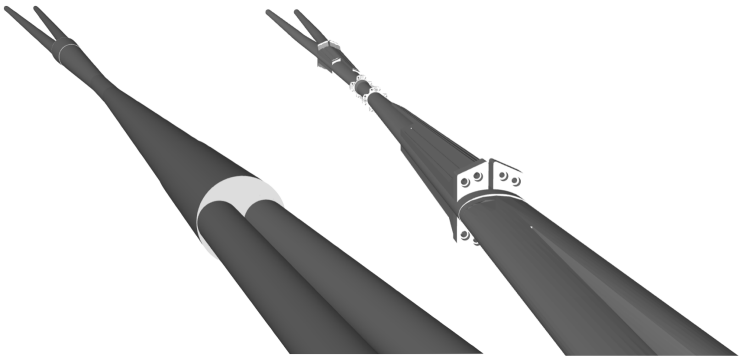


Fig. 11: Shape based (left) and CAD (right) models of the IR beam pipe, as rendered by the TGeo based visualisation tool using `JSROOT`, with focus on the beam pipe separation region.

method can assign only one material to a mesh, so different `.stl` files are exported for each material used. In the current version the following elements are included:

- The AlBeMet162 engineered model of the double layered central chamber and the elliptoconical chambers, with their cooling manifolds and inlets/outlets
- The low impedance Copper beam pipe separation region and independent beam pipes
- The 5 μm gold layer coating inside the central chamber
- The paraffin cooling layer for the central chamber
- The water cooling for the elliptoconical chambers
- The two Synchrotron Radiation Tungsten masks placed upstream just after the innermost final focus quadrupole

As the engineered design progresses, the present elements will be modified accordingly and new ones will be added, like the remote vacuum connection and the bellows.

4.2 Vertex detectors

4.2.1 Double-layer vertex detector

The 'double-layer silicon' vertex detector is the vertex detector evolved from CLICdet [53] which is used in CLD. The layout is based on double layers, that is, two sensitive layers fixed on a common support structure. The barrel includes multiple layers, while the forward region is covered by several sets of discs on both sides of the barrel. The inner geometry is determined by MDI constraints, while the rest of the layers are scaled down the CLICdet layout. In a forthcoming engineering study, such layout details will have to be revised. Unlike CLICdet, the CLD vertex detector in the forward region uses planar discs instead of spirals. The spiral design in CLICdet was optimised for air flow, but air cooling isn't considered adequate for the CLD vertex detector. Further details about this vertex detector can be found in [54], Section 3.

The geometry to be simulated is built by 3 detector drivers. The detector barrel is constructed using the `ZPlanarTracker` driver, which reads the position and composition of each layer from the corresponding compact file. Currently, both the support structure and sensors are made of silicon. The detector end-cap is built using the `VertexEndcap_o1_v06` driver. According to the detector section in the compact file, 6 layers are constructed from 6 different modules. Each module defines a "petal" of the disk, consisting of a sensitive volume and a support volume, both of which are currently made of silicon. Additionally, a 0.02 mm thick copper disk, constructed by the `TrackerEndcapSupport_o1_v02` driver, is included to account for the cabling. This driver does not define individual pixel volumes and is agnostic of the spatial resolution, which is defined at the digitisation level.

4.2.2 Single-layer vertex detector

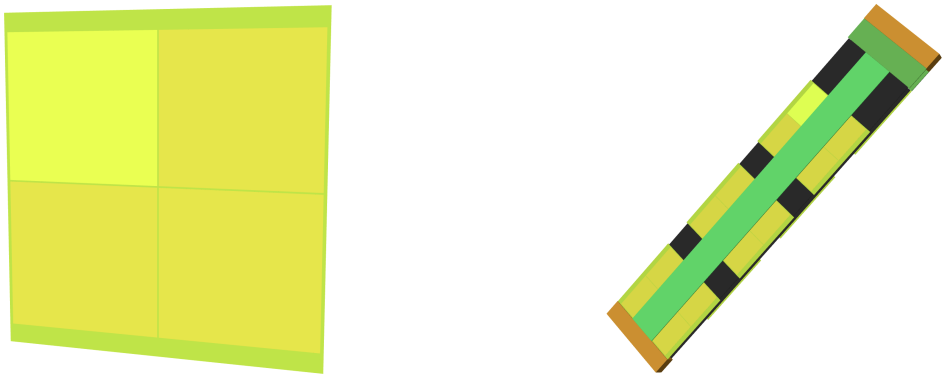
The second vertex detector layout for FCC-ee was proposed for the IDEA detector concept and is also used for ALLEGRO. It was previously implemented in native Geant4, enabling full simulation of the detector [55, 56]. The native Geant4 version is now deprecated and only the DD4hep implementation is further maintained.

The DD4hep description of this vertex detector [57] uses single-hit layers of MAPS and features an accurate representation of the first engineered vertex detector for FCC-ee. It is integrated into the machine-detector interface (MDI) region. The single-layer vertex detector consists of an inner vertex barrel detector with three layers and an outer vertex with two barrel layers and three disks per side. For a thorough discussion of the single-layer vertex detector design and the MDI region please refer to the MDI note of the FCC Feasibility Study Report and [51, 58, 59].

The single-layer vertex detector is assembled using two newly written builders hosted in k4geo (VertexBarrel_detailed_o1_v02_geo.cpp and VertexEndcap_detailed_o1_v02_geo.cpp) for the barrels and the disks respectively. The detector is then constructed using the dimensions of the various components as defined in the compact file. The detector builders are highly generic – they can be used to describe all kinds of semiconductor-based tracking detectors accurately.

The sensors are described by multiple cuboids with a given thickness. Each element can either be sensitive or insensitive, which allows one to accurately describe the insensitive sensor peripheries. In the case of the ARCADIA [60] sensor used in the inner vertex barrel, for example, 2 mm out of the sensor width of 8.4 mm are periphery and thus will not generate hits when particles traverse through there. The ATLASPix3 MAPS [61] used in the outer vertex forms quad modules, where both the peripheries of the individual front-ends and also the gaps in-between them are correctly described (see Fig. 12). In the same manner as for the double-layer silicon drivers, this driver does not define individual pixel volumes and is agnostic of the spatial resolution, which is defined at the digitisation level.

The sensors are on top of support structures (staves) that go along z (barrels) or are placed at different r and ϕ to form petals (disks). The support structures, readout cables and cooling pipes that go along the staves or sit at their ends (end-of-stave structures) can be added by defining their width, length, thickness, material, and so



(a) ATLASPix3 quad module with four front-ends.

(b) Stave with three ATLASPix3 modules on both sides

Fig. 12: Visualisation of outer vertex ATLASPix3 quad sensor (Fig. 12a) and a disk stave (Fig. 12b). One ATLASPix3 front-end is highlighted, sensitive regions are in yellow, insensitive in light green.

on. A stave object is defined by stating all associated support structures and readout cables.

The vertex detector envelope is placed into the world volume. It has two cone cut-outs to make space for the forward beam instrumentation and a cylinder cut out in the middle for the central beam pipe. As shown in Fig. 13, the barrel and endcap detectors are *assemblies* placed into the vertex envelope. They consist of cylindrical mother volumes, one for each barrel and disk layer. Stave assemblies of all the sensor, support and readout volumes are placed into these mother volumes according to the number of staves and their position as defined in the XML file. Optionally, a mother volume for each stave can be defined as well, providing a faster detector construction in DD4hep. In the vertex detector this feature is not used as these stave mother volumes would clash with each over, generating unwanted overlaps.

The complete vertex detector geometry is shown in Fig. 14. Some more geometrically complex structures are described by proxy volumes or directly imported using DDCAD [62]. In the inner vertex barrel, the stave holder structure is described by boxes of carbon fibre and Rohacell, with a reduced density (66% of nominal) to account for the lightweight mode of construction. The conical support of the inner vertex barrel, that sits on top of the beam pipe, and the cooling cones in the forward direction are imported from CAD. For the vertex outer barrel, the material budget contribution of the light-weight truss structure giving structural integrity to the staves is taken into account by a proxy layer of carbon fibre. The geometry description also contains placeholder structures to represent the cooling pipes that go along the stave and the contained water. The global disk support structures are approximated by proxy cylindrical volumes with thicknesses of $\approx 2.5\,\text{mm}$ for each disk layer (6 mm in CAD design, but with holes). The material budget and performance of the single-layer vertex detector is discussed in the MDI note of the Feasibility Study Final Report.

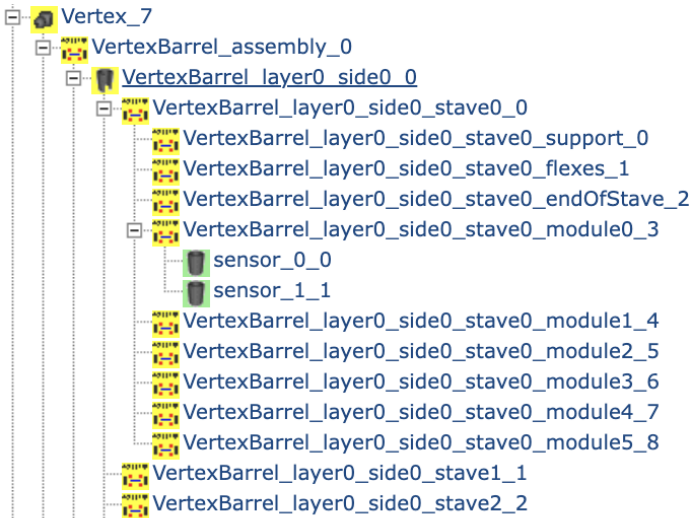


Fig. 13: Hierarchy of volumes in inner vertex.

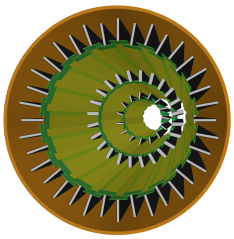
4.2.3 Ultralight curved vertex detector

A third vertex detector concept has been studied [57, 63] which is based on the ALICE ITS3 developments [64]. The three layers of inner vertex from Section 4.2.2 are replaced by four layers of wafer-scale curved MAPS, so that only very few support structures and almost no cables are needed within the acceptance of the vertex detector. The outer vertex, however, remains unchanged with respect to Section 4.2.2. Details on this vertex detector concept can be found in the MDI note of the Feasibility Study Final Report.

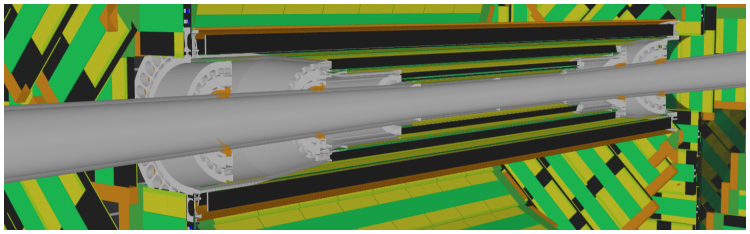
The full simulation study of this concept started in late 2023 in order to validate a first Delphes study [57, 63] and to advance from an idea for an advanced vertex detector to a concept.

To describe the ultralight vertex detector concept in DD4hep, the vertex barrel constructor from Section 4.2.2 (VertexBarrel_detailed_o1_v02_geo.cpp) was extended to also allow the placement of curved structures. Instead of `dd4hep::Box`, `dd4hep::Tube` volumes are used to describe cylindrical sectors both for the sensors and for support and readout structures that are bent. Fig. 15 shows the DD4hep implementation of one curved sensor. To let Key4hep correctly handle hits in these curved sensors, the cylinder sensitive element in DD4hep was extended to also allow cylindrical sectors instead of complete cylinders only.

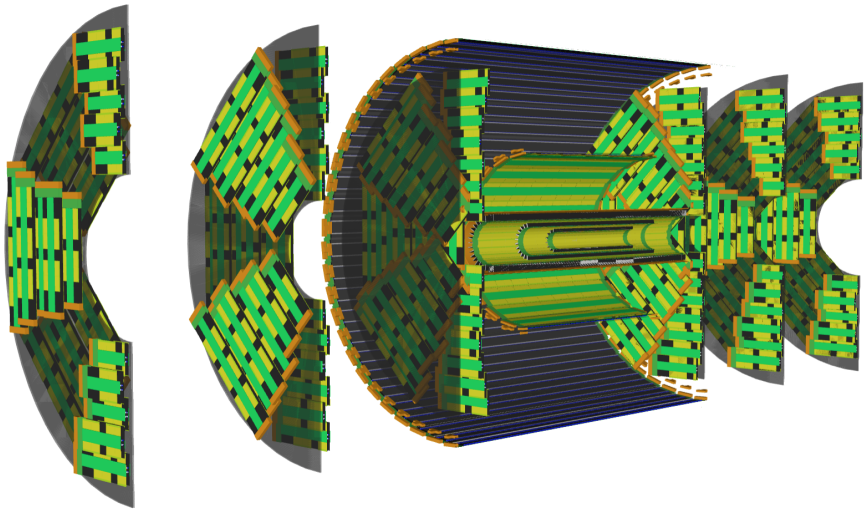
The DD4hep description of the ultralight inner vertex features large sensors with a thickness of 50 μm of silicon. To include the material coming from the metal layer on top of the wafer-scale sensor (providing power), an additional 16 μm of silicon are added. Since such thin curved layers on top of each other can create volume overlaps, just one layer of 66 μm of silicon is considered. These overlaps of curved structures also make it necessary to add a small clearance of 33 μm in-between any two curved structures.



(a) Inner vertex barrel without inner support.



(b) Inner vertex barrel with CAD-imported support and beam pipe.



(c) Complete single-layer vertex.

Fig. 14: Visualisation of the single-layer vertex detector in DD4hep.

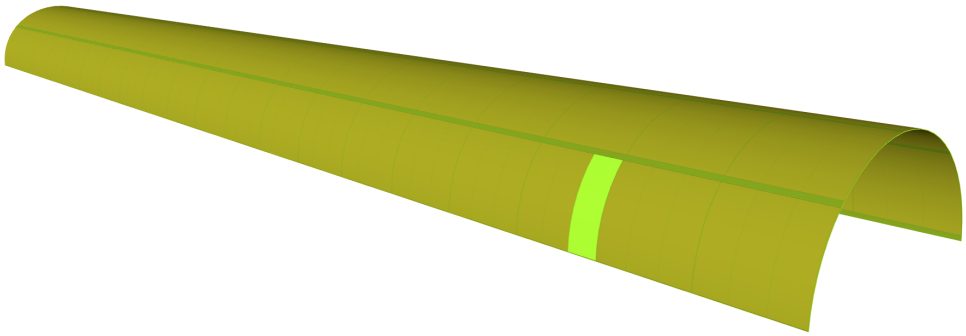


Fig. 15: Curved wafer-scale sensor forming half of the first inner vertex barrel layer. One sensitive element is highlighted in light green. The other sensitive elements are in yellow while the insensitive areas are in dark green.

The model also includes the carbon-foam mechanical supports between the half-barrel layers of curved silicon sensors and also the flex circuits at the ends of the sensors. Figure 16 shows the first layer in the DD4hep model that covers $\theta > 125\,\text{mrad}$ ($\cos(\theta) < 0.992$). Inside this acceptance, due to the insensitive peripheries in the sensor and the gap of 1.25 mm between the two half-cylindrical sensors, the sensitive fraction adds up to $\approx 91\%$ for the first layer.

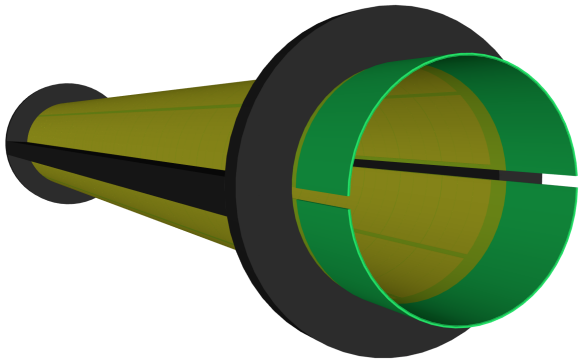
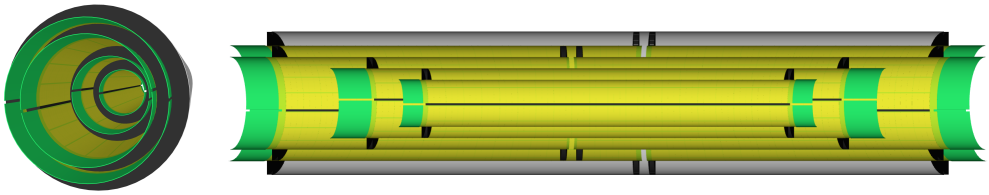


Fig. 16: The first ultralight inner vertex barrel layer.

The second layer has the maximal length possible (13 repeated sensor units, RSUs), due to the limitation in wafer size of 12 inch, and features three RSUs in ϕ (two in first layer). It thus has a coverage down to θ of 143.5 mrad ($\cos(\theta) < 0.99$).

The third and fourth layers consist of two sensors in a row in z per half-barrel to cover the forward region sufficiently, reaching 135 and 167 mrad respectively. They feature an asymmetrical design in z so that the gap between the two sensors of one layer is covered by the next. The impact on the hermeticity through the gaps between the two half-barrels and the sensor peripheries in the layers is mitigated by rotating the layers in ϕ one with respect to the other, as shown in Fig. 17.



(a) Front view. (b) Cut open side view

Fig. 17: The complete ultralight vertex detector. At least three hits are ensured by rotating the layers in ϕ one with respect to the other (a) and by an asymmetric design of the third and fourth layer (b).

The DD4hep model described in this section still misses the air-cooling structures and the off-detector services, which both, however, are in the forward region and mostly outside of acceptance of the vertex detector. Further details on the ultralight vertex detector concept, such as the material budget plots, can be found in the MDI Feasibility Study Final Report.

The presented detector constructor allows one to build hybrid vertex detector models, where some layers are curved and others using overlapping staves. Mixed solutions of the vertex detectors presented in Section 4.2.2 and Section 4.2.3 could be interesting if the challenges e.g. on hit rate cannot be dealt with in the first layer of a curved detector or if a silicon tracker is to use also curved sensors, as e.g. planned for ALICE 3 [65].

4.3 Trackers

4.3.1 Full stereo drift chamber

This detector is part of the central tracking system, aligned with the solenoid's magnetic field. It is designed for accurate tracking, precise momentum measurement, and particle identification. A key feature of this drift chamber is its high transparency, with 1.2% radiation length in the radial direction and 5% in the forward (endcap) direction [66].

The detector consists of a 4-meter-long cylindrical vessel that spans a radial space from 0.35 to 2 meters and is filled with a gas mixture of 90% helium and 10% isobutane. The vessel wall is made of a sandwich of carbon fibre and PE foam to meet material budget requirements. In the final detector, most of the material budget will come from the wire support structures and electronics, which are accounted for by tuning the thickness and radiation length of the vessel endcap wall. During Geant4 simulations, the mean excitation energy (MEE) of the gas mixture is set to 48 eV, replacing the value that Geant4 calculates.

A drift chamber is a gaseous detector which is able to measure the electrons drift time to determine the spatial coordinates of an ionizing event. In its basic form, a single-cell drift chamber consists of a electron drift region, where the creation of ion-electron pairs happens and electrons are drifted by a (ideally uniform) moderate electric field, and an avalanche region, in the immediate vicinity of the anode (sense) wire. In addition, the amount of electrons and its time distribution corresponding to each ionizing event is used to determine the nature of the incident particle [67].

The detector contains 56,448 drift cells arranged in 112 layers [66]. The global geometry is twisted by 30 $^\circ$, making the layers hyperboloid-shaped and the cells twisted tubes. Each cell is placed within the corresponding layer around the Z axis as many times as needed to fill the entire layer volume. Layers and cells are represented in the geometry to ensure that particles make at least one step within each cell they cross during the simulation without needing extra step limitations. The detector driver can be found in the central k4geo repository [68]. The twisted tube shape is represented internally as intersection of surfaces [69], leading to slow navigation and long simulation times (around 1 second per $H \to q\bar{q}$ event). The performance will be improved by replacing the geometrical segmentation resulted from the use of twisted tubes by a

virtual (DD4hep) segmentation for each layer. This virtual segmentation requires to limit the step within the region of the drift chamber.

The cross section of a cell is shown in the left plot of Fig. 18. A 20 μm thick sense wire is placed at the centre of the cell, and $40 - 50$ μm thick field wires at the corners. The position of the wires is slightly modified from the original design to fit entirely inside each cell. This means each cell contains a sense (anode) wire and five field (cathode) wires, all with the same twisted angle (though the stereo angle varies with radial distance, the twisted angle is global). Note that cells of consecutive layers are placed with a shift along ϕ of 25% of the cell width, manifested by the position of the field wires. The right plot of Fig. 18 shows the cross section of the drift chamber at $Z = 0$ where only the sense wires are shown for simple visualisation. The original design [66] describes two layers of guard wires (2016 in total), which will be added in a future revision of the geometry but their expected contribution to the total material budget is negligible.

The large number of wires and the twisted geometry of the drift cells make this detector complex to implement. The length and position of every wire, derived from the global parameters, are calculated and stored by an object of a new C++ class which has been added to the central DD4hep repository for common use [70]. This class includes many functions useful for building the geometry and later track reconstruction, such as `Get_cell_phi_angle` which return the rotation angle ϕ around the Z axis of a given cell or `Get_phi_width` which return the cell width along ϕ. DD4hep provides a data-extension feature that allows a custom C++ class object to be attached to a DD4hep Detector Element. This feature is used for the drift chamber, so the object that stores the wire and cell parameters is attached to the detector, keeping the geometry consistent with the implementation.

In the current implementation of the geometry, the field wires and sense wires (anodes) are present but no electric field neither electron drift nor avanlanche is simulated by Geant4. Instead, a parametric model, derived from Garfield++ simulations, provides the cluster size and number [71] during the digitisation step of drift chamber events.

4.3.2 Silicon tracker

The all-silicon tracker detector evolved from the CLICdet [53], aiming for a similar momentum resolution with a lower magnetic field. As in case of the double layer silicon vertex sub-detector (Section 4.2.1), it was developed to be used by FCC-ee experiments, particularly CLD. It is divided between an inner and outer tracker systems, each one made by layers grouped into a barrel and closing end-caps. Further details about this tracker detector can be found in [54], Section 4.

The geometry to be simulated is built using two detector drivers for the sensors and the electronics, and another three for the support structures. The detector barrel for both the inner and outer trackers is constructed with the `TrackerBarrel_o1_v05` driver. This driver arranges square base modules around the cylindrical surface by placing them along the ϕ and z axes. The composition of each base module is detailed in the compact file, specifying materials such as silicon for the sensor, carbon fibre, Kapton, epoxy, and water for the structure and cooling.

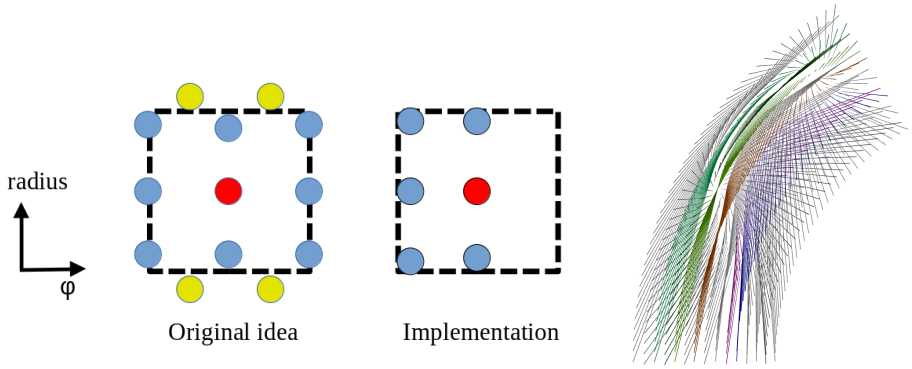


Fig. 18: The plot on the left shows the cross section a drift cell, where the dash black line correspond to the border of the cell, blue circle to the field wires, red circle to the sense wires, and yellow circles to the field wires of the next/previous layer cell. The thickness of the wires is exaggerated for visualisation purposes. The plot on the right shows a cut view at $Z = 0$ of the sense wires (placed at the centre of the drift cell) of $\Delta\phi = 60^{\circ}$ and the 15 first layers, each colour corresponds to one layer.

The end-caps of the inner and outer trackers are constructed using the `TrackerEndcap_o2_v06` driver. This driver positions petals around the ϕ axis, with consecutive petals slightly overlapping at their edges by being placed at alternating Z positions. Each petal has a trapezoidal shape and is made up of rectangular modules named texttttInnerTrackerEndcapModule_AxB_Out, where A and B indicate the number of square chips, each measuring 15.1 $\times$ 15.1 mm^2 (30.1 $\times$ 30.1 mm^2), that are combined to form the module for the inner tracker (outer tracker). The module definitions are independent of the detector definition, allowing them to be used across multiple sub-detectors.

The support structures are created using several detector drivers: `TrackerBarrelSupport_o1_v01`, `TrackerEndcapSupport_o1_v02`, `TubeSupport_o1_v01`. These drivers build simple layered structures in the shape of disks or barrels, according to a list of slices of different materials and thickness specified in the compact file.

4.3.2.1 How to resize the outer geometry

The outer geometry may need to be reduced to accommodate other sub-detectors. The following (independent) modifications are required for the tracker:

- Outer Tracker Barrel: The space between layers is reduced proportionally by decreasing the radius of each layer. The radius should be adjusted to fit an integer number of modules per ring, ensuring they fit without overlaps or gaps. The size of the modules used to build the barrel is unchanged.
- Outer Tracker End-Cap: Two main changes are needed: first, move the disks closer to $z = 0$, reducing the space between them proportionally; second, replace the outer radial module in the petal with one that has a smaller y-size.

- Outer Tracker Envelope: Reduce the overall radius and length of the outer tracker. These parameters will guide the resizing of the passive structures.
- Inner Tracker End-Cap: Similar to the outer tracker, the space between layers is reduced by moving the disks closer to $z = 0$.

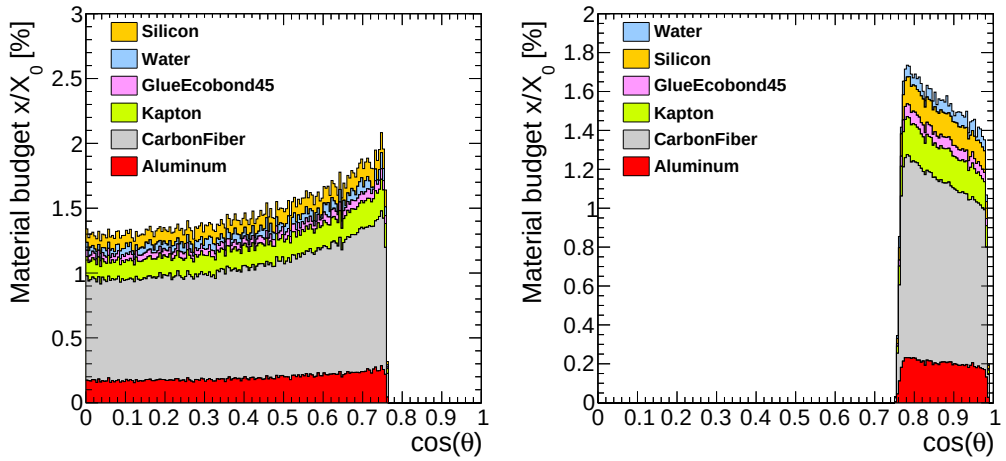
4.3.3 Silicon wrapper

A silicon wrapper detector can supplement a gaseous tracker by providing a last precise measurement at large radius. The time resolution of $\sim 30\,\text{ps}$ would also complement the $\mathrm{d}N_{\text{clusters}}/\mathrm{d}x$ information to improve particle identification. In the case of the IDEA detector concept, the silicon wrapper is surrounding the drift chamber ($z \leq \pm 2.25\,\text{m}$ and $r < 2\,\text{m}$), leading to an area of more than $100\,\text{m}^2$.

The silicon wrapper consists of two barrel layers and two disks per side that are overlapping at their interface to provide full hermeticity (with the exception of sensor periphery regions) down to $\cos(\theta) \lesssim 0.989$ (θ of 151 mrad). The target is to have at least two hits for most tracks.

Since the silicon wrapper uses a geometry similar to the vertex detector (barrel layers and disks of silicon sensors), the same constructor algorithms are used (see Section 4.2.2).

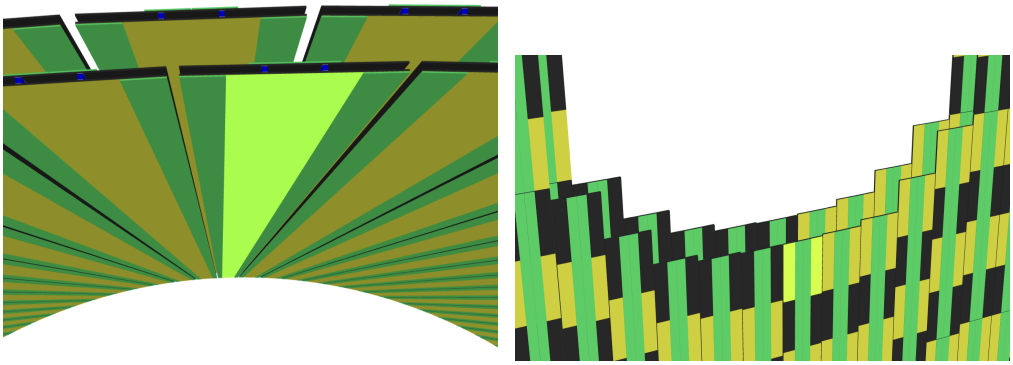
The barrel layers (disks) are made up of staves (modules) with sensors on both sides, flexes on top and a support and cooling structure in the middle. The sensors have a thickness of $50\,\mu\text{m}$. Each stave/module has a 1.4 mm carbon fibre support, the same cooling pipes and four readout flexes as for the vertex detector described in Section 4.2.2. Overall, this silicon wrapper implementation features a material budget of $\approx 0.7\%$ of X_0 per barrel layer/disk as shown in Fig. 19.



(a) Barrel (b) Disks

Fig. 19: Material budget in the DD4hep silicon wrapper implementation for the two barrel layers (a) and the two disks per side (b).

Given the amount of silicon sensors needed to cover the large area ($\mathcal{O}(20,000)$ for the barrel and $\mathcal{O}(15,000)$ for the disks), it is not possible to have the same level of detail in the detector model as e.g. in the vertex detector without compromising on the time needed to initialise the detector simulation (too many volumes per node in the geometry hierarchy). The smaller importance of accurate material budget estimation far away from the IP means that the impact on the simulation is negligible, however. In the barrel, the sensors are thus described by just one sensitive element along the whole stave. For the disks, instead of having modules with 3, 6 and 12 sensors, each module is just featuring 1, 2 or 4 larger sensors. In this way, only 604 (5056) sensitive elements are needed for the silicon wrapper barrel (disks). Fig. 20 shows the silicon wrapper DD4hep model currently implemented using two barrel layers and two disks per side.



(a) Two layers of barrel sensors.

(b) Two disks per side.

Fig. 20: Close view of the silicon wrapper. One sensor piece is highlighted in light green.

An updated version of the silicon wrapper XML file is being developed and targets only ≥ 1 hit leading to a reduced amount of sensors needed.

4.4 Array of RICH cells (ARC)

The 'array of RICH cells' (ARC) sub-detector provides PID of charged particles covering the momentum range of 1 to 0 GeV/c, with a small radial extent (∼20 cm) and a material budget of 5% on average. Charged particles that pass through the radiators, with momentum above the threshold for Cherenkov radiation, emit photons that are focused onto a photon sensor by a spherical mirror. From the hit position of the photon on the photon sensor, together with knowledge of the charged track trajectory, the Cherenkov angle is reconstructed [72].

The light sensors in earlier RICH detectors, such as the ones used in DELPHI [73] and SLD, required a tremendous size of gas radiator to work. Modern optical photon sensors such as SiPM have better performance, thus requiring thinner Cherenkov radiators, leading to smaller detectors.

The global size of the ARC corresponds to a cylinder with an outer radius of 2 meters, 4.4 meters long, and thickness of 20 cm. The vessel wall is 1 cm thick, made of a sandwich of Polyethylene (PE) foam encapsulated between two layers of carbon fibre 1 mm thick. The RICH cells are grouped into a barrel and two endcaps. One of the endcap results from 180° rotation around the X axis (mirror reflected geometry requires more memory). The barrel geometry results from rows of 18 unique RICH cells, shown in Fig. 21, placed 27 times around Z axis. Each endcap is made of 6 sectors of 21 unique RICH cells, as shown in Fig. 22. The gap between the endcap and the barrel has to be filled with partial cells, which are not present in the current geometry implementation. The full geometry of the ARC is shown in Fig. 23, where the vessel wall and Cherenkov radiators are invisible for the sake of simplicity. The same figure shows the material scan of the current implementation of the detector. The spikes at $\theta = 40^\circ$ and $\theta = 140^\circ$, mainly due to the wall, correspond to the transition between the endcap and the barrel. This feature will be investigated in the engineering design phase at a later stage to determine if we can avoid the duplication of material.

Each of the RICH cells in Fig. 21 are different due to their different location relative to the interaction point (IP). In particular, the positions of the photon sensor and the mirror, as well as the mirror radius of curvature, must be optimised for an optimal PID performance. The optimisation is performed using a dedicated standalone program that performs a ray-tracing simulation of charged tracks and photons. For each charged track, its emitted photons are ray-traced. Their Cherenkov angles are reconstructed [72] and the Cherenkov angle resolution,

$$\Delta\theta = \frac{1}{\sqrt{N}} \times \frac{1}{N-1} \sum_{i=1}^{N} (\theta_i - \langle\theta\rangle)^2, \tag{4}$$

is calculated. In Eq. (4), the angle θ_i is the reconstructed Cherenkov angle of each individual photon, while $\langle\theta\rangle$ is the average Cherenkov angle. Effectively, Eq. (4) is the single photon Cherenkov angle resolution, divided by $\sqrt{N}$ to obtain the achieved resolution with N detected photons.

To evaluate the Cherenkov angle resolution with Eq. (4), an ensemble of 2×10^4 random charged tracks are simulated for each cell. The cells are optimised by allowing the curvature, vertical position and horizontal position of the mirror to vary. Furthermore, the horizontal position of the photon sensor and its tilt angle are also varied, which makes a total of five free parameters for each cell. Since there is a finite number of photons used to calculate $\Delta\theta$, a stochastic optimisation, known as Differential Evolution [74], is employed to find the optimum geometry that minimises $\Delta\theta$.

The geometry can be scaled globally and the optimisation would still be valid, but in general a new optimisation must be done for a different global geometry. The different colours of the spherical mirrors identify the different unique cells.

The shape of the cells corresponds to an hexagonal prism with a distance from side to side of 26 cm. The barrel cells are semi-projected in the xy-plane, and a dedicated tessellated solid was used to define the shape of these cells. Each cell is made by a gaseous Cherenkov radiator, and all of them are placed in a global volume made of the same gas. Segmenting the global gas volume into cells is a trick convenient for a well balance volume tree, leading to an optimal navigation of the geometry during the full simulation. The actual detector may not have such segmentation of the global gas into independent cells. The rest of the cell components are placed inside this volume: a mirror, which focuses the Cherenkov photons into a SiPM as light sensor of size 8×8 cm^2; and an aerogel layer 1 cm thick as second Cherenkov radiator to cover the

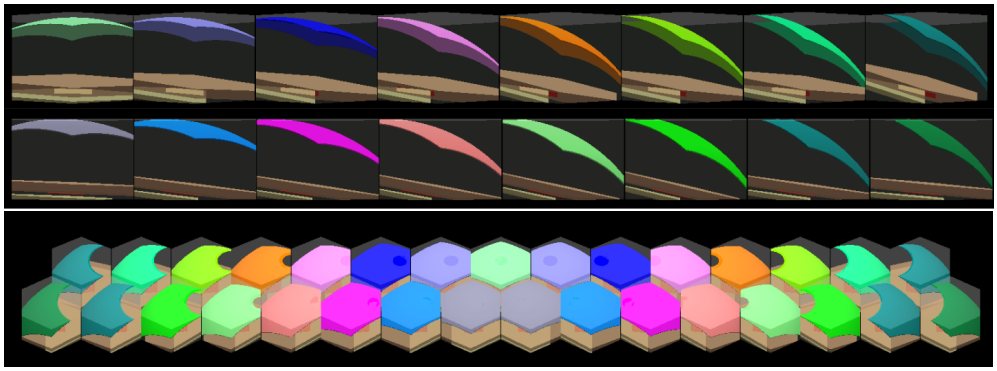


Fig. 21: Sagittal view (top) and top view (bottom) of the row of cells that make up the barrel of the ARC. The ARC barrel geometry results from placing 27 times the row of cells shown in the bottom picture around the Z axis. The spherical mirror of each cell is positioned to face the IP by off-centring sphere cap with respect to the cell. The colour of the mirror identifies different unique cells.

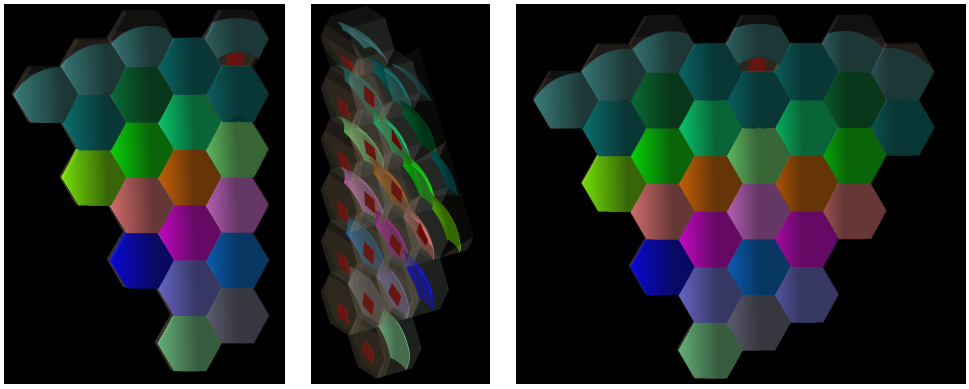


Fig. 22: Endcap partial sector (left and centre), which is mirrored to make a full sector (right). This sector is later repeated 6 times around the z-axis to make up the full endcap. The colour of the mirror identifies different cells.

range of momentum below 10 Gev/c. The aerogel tile is placed on top of the SiPM. The SiPM will require some structure beneath for cooling, thus a dummy plate is introduced in the simulation to account for the dead material. Both the cooling plate and the spherical mirror are made of carbon fiber to minimise the material budget. Figure 24 shows the cross section of a RICH cell for the ARC. The implemented geometry defines the mirror shape as intersection a spherical cap with the cell shape, in such a manner that the mirror fits perfectly within the cell volume. This trick simplifies the geometry tree and thus speeds up tracking of particles during the full simulation because all the cell components are placed within one volume. If the mirror shape is defined as spherical caps, it would protrude the cell shape forcing to place all the internal components at the same level (the global gas volume for example). The detector driver that build this geometry is called ARC_geo_o1_v01, hosted at the k4geo GitHub repository [75].

The material chosen as gas radiator is C_4F_{10} at 1 atm, which gives good momentum range for kaon-pi separation with acceptable photon yield for the parameters assumed. Xenon pressurised at 2 atm may provide a similar performance if fluorocarbons are unacceptable, at the cost of increasing the material budget.

The Geant4 full simulation reads the refractive index associated to the radiator materials in order to create the Cherenkov (optical) photons and also transport them across the gas and the aerogel. Rayleigh scattering in the aerogel is simulated according to the Rayleigh scattering attenuation length data table which is calculated from a clarity factor of 0.005 $\mu m^4/cm$. Optical absorption in the radiators is disregarded at the moment. The DRD4 collaboration is investigating new materials to be used as

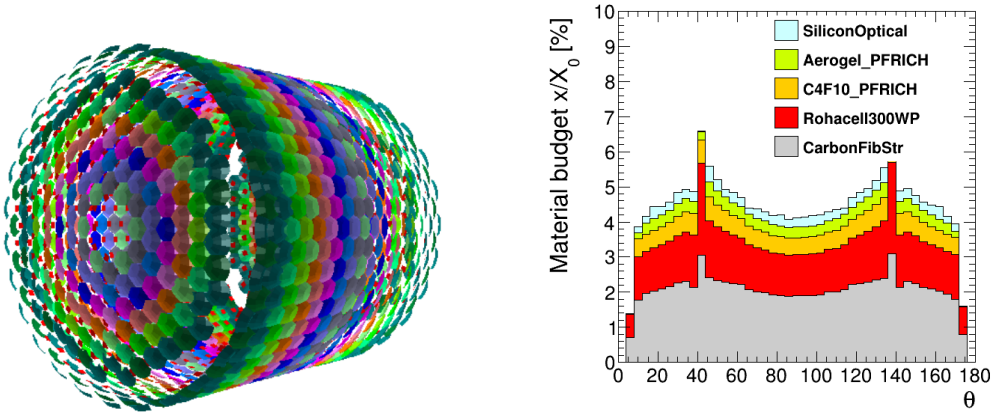


Fig. 23: Left: ARC mirrors and light sensors placed according to the optimised geometry for this detector size. Unique cells within the barrel or endcap are identified by different mirror colours. The light sensors corresponds to the small red squared-like volumes. Right: material scan, averaged over ϕ, bin width of 4°. The walls are mainly made of the PE foam (*Rohacell300WP*), sandwiched by carbon fiber (*CarbonFibStr*). The materials named *C4F10_PFRICH* and *Aerogel_PFRICH* correspond to the Cherenkov radiators, and *SiliconOptical* corresponds to the material of the SiPM.

Cherenkov radiators, and the optical properties of these materials can replace the current ones to assess the performance of the detector before building a prototype. The mirror reflectiveness comes from a thin metallic coating, which is disregarded in the geometry description, although its optical reflectiveness parameter is defined to be 100%, thus optical photons are reflected without losses.

4.5 Calorimeters

4.5.1 Luminosity calorimeter

The driver named `LumiCal_o1_v01` is used to create the Luminosity Calorimeter (LumiCal) for every detector concept. This driver implements the same geometry developed for the ILD [76]. LumiCal is a sampling calorimeter with a cylindrical shape, where layers are stacked along the Z-axis. The compact file specifies the parameters that define the material of the slices that make up each layer, the segmentation (defined within the readout), position, and crossing angle (which is used by the driver to rotate the detector). For consistency, the same driver is used to create ancillary services such as the back shield and cooling systems.

4.5.2 Noble liquid calorimeters

Initially proposed for FCC-hh due to the intrinsic radiation hardness of liquid argon and its excellent resolution and linearity over a large energy range, a noble liquid electromagnetic calorimeter design has then been adapted to the FCC-ee running scenario. The proposed detector concept consists of a barrel section in the central region and of two end-caps in the forward regions. Both sub-detectors have been implemented in the FCC-ee full simulation using the DD4hep [1] toolkit. Their geometries are described in the following paragraphs.

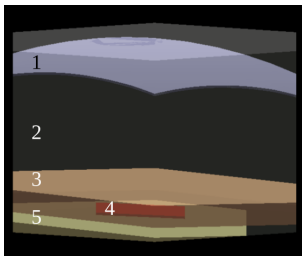


Fig. 24: Geometry of a RICH cell for the ARC sub-detector. The Cherenkov photons are produced in the radiator gas (2) and the aerogel (3), reflected by the spherical mirror (1) and detected by the light sensor (4). A dummy piece (5) is added to account for the cooling plate of the sensor. Cherenkov radiation is forward with respect to the charge particle trajectory, so the mirror is placed in the outer part of the cell.

4.5.2.1 Barrel

The barrel electromagnetic noble liquid calorimeter is a sampling detector using liquid argon as active medium and lead/steel absorbers constituting the main passive material. The absorbers and electrodes are straight, and inclined in the $R - \phi$ plane with respect to the radial direction, as shown in Fig. 25.

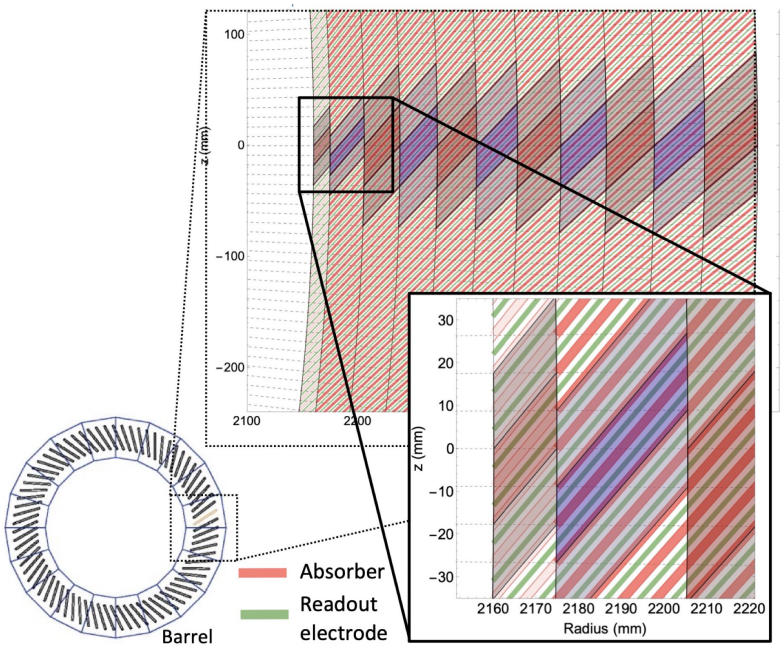


Fig. 25: Drawing of the barrel structure of the noble liquid electromagnetic calorimeter

The detector has both transverse and longitudinal segmentation, allowing the reconstruction of the direction of impinging particles. Alternative design options include liquid krypton and tungsten for the active and passive material, as well as trapezoidal absorbers to keep the sampling fraction constant among longitudinal layers and further improve the constant term of the energy resolution. The code developed for the simulation can also handle such alternative scenarios, which can be generated by replacing the default settings of the simulation (defined in XML configuration files) with custom settings.

The geometry of the barrel detector is built by the C++ driver `ECalBarrel_NobleLiquid_InclinedTrapezoids_o1_v03_geo` from the `k4geo` repository [50] based on the properties (geometry, material, readout granularity, ...) specified in XML files in e.g. the `FCCee/ALLEGRO/compact/ALLEGRO_o1_v03` folder. The various geometrical parameters of the calorimeter, such as the dimensions of the cryostat and calorimeter bath, absorbers, electrodes, number of absorbers, number of layers, readout granularity, as well as the various materials used for the active and passive media,

can be configured by changing the corresponding values in the aforementioned XML file.

The calorimeter is located inside a cryostat, which has the shape of a hollow cylinder, centred at the interaction point and with axis along the z-direction. The cryostat also holds the main solenoid that provides the magnetic field for the main tracking sub-detector. The calorimeter and its cryostat are located within an envelope, whose parameters are defined in `DectDimensions.xml`:

- `BarECal_rmin`: the inner radius
- `BarECal_rmax`: the outer radius
- `BarECal_dz`: half of the length along z

The envelope is a hollow cylinder (`ddhep::tube`) filled by default with air.

The calorimeter itself is filled with a bath of active medium, i.e. a liquefied noble gas. In the bath are immersed a certain number of absorber planes, which are inclined in azimuth from the radial direction by a given angle. The section of the absorber planes is an isosceles trapezoid, reducing to a rectangle in the special case in which the two parallel sides have the same length. Each absorber is composed of a sandwich of two outer thin sheets glued to an inner thicker core. The space between the absorber is divided in two equally-thick gaps by the electrode, which in the ALLEGRO detector concept is provided by a multi-layer PCB.

The electrodes are segmented longitudinally into a given number of layers. In the simulation, for the derivation of a per-layer sampling fraction, also the absorbers and the LAr bath have the same segmentation, each layer of the barrel calorimeter corresponding to a separate physical volume. The first layer is used as a presampler. To achieve a ϕ-uniform response of that layer and increase its sampling fraction, the inner core material of the absorbers is replaced with the noble liquid in the presampler layer.

The parameters describing the geometry and the materials of the detector are specified in the `ECalBarrel_thetamodulemerged.xml` configuration file.

Some constants used to calculate the geometry are included in the `define` block:

- `AirMarginThickness`: air gap between cryostat vessels
- `CryoBarrelFrontWarm`, `CryoBarrelFrontCold`: the thickness of the inner face of the cryostat vessel (warm and cold sections), from which the total `CryoBarrelFront` is calculated
- `CryoBarrelBackWarm`, `CryoBarrelBackCold`: the thickness of the outer face of the cryostat vessel (warm and cold sections), from which the total `CryoBarrelBack` is calculated
- `CryoBarrelSideWarm`, `CryoBarrelSideCold`: the thickness of the sides of the cryostat vessel (warm and cold sections), from which the total `CryoBarrelSide` is calculated
- `NLiqBathThicknessFront` and `NLiqBathThicknessBack`, the thickness of the liquid nitrogen bath used to cool the cryostat cold side
- `BarCryoECal_rmin`: equal to `BarECal_rmin`+`AirMarginThickness`
- `BarCryoECal_rmax`: equal to `BarECal_rmax`-`AirMarginThickness`
- `BarCryoECal_dz`: equal to `BarECal_dz`

- EMBarrel_rmin = BarCryoECal_rmin+CryoBarrelFront+NLiqBathThicknessFront, EMBarrel_rmax = BarCryoECal_rmax-CryoBarrelBack-NLiqBathThicknessBack and EMBarrel_dz = BarECal_dz-CryoBarrelSide: the calorimeter active volume
- Bath_rmin=EMBarrel_rmin - safeMargin, Bath_rmax=EMBarrel_rmax + safeMargin, the active medium bath inner and outer radii with a safety margin (parameter SafetyMargin) for the inclination of the electrodes and absorbers.
- Pb_thickness: the thickness of the core part of the absorber, at the inner radius of the calorimeter
- Pb_thickness_max: the thickness of the core part of the absorber, at the outer radius of the calorimeter. Same as Pb_thickness for parallelepiped-shaped absorbers, larger than that for trapezoidal-shaped absorbers.
- InclinationAngle: the inclination angle of the lead plates in $R - \phi$
- Sensitive_thickness: the thickness of the active volume between two absorber planes at the barrel inner radius, perpendicular to the readout plate
- ECalBarrelNumPlanes: number of electrode planes
- ECalBarrelNumLayers: number of longitudinal layers
- planeLength: length of the electrode, calculated from EMBarrel_rmin, EMBarrel_rmax and InclinationAngle
- Steel_thickness: total thickness of the two sheets glued outside of the absorber
- Glue_thickness: total amount of glue in one passive plate
- readout_thickness: thickness of the electrode

The detector properties are specified in the detector tag in the XML file. Its sensitive type is SimpleCalorimeterSD, which is later interpreted in ddsim as calorimeter. The dimensions, rmin, rmax and dz, are set equal to BarCryoECal_rmin, BarCryoECal_rmax, and BarECal_dz.

In terms of geometry tree, the cryostat is at the same level as the noble liquid bath. It is implemented in the C++ driver as three separate volumes: two dd4hep::Tube representing the inner and outer walls, and a dd4hep::SubtractionSolid between the cylinder within the walls and the inner bath, representing the sides of the cryostat. The cryostat properties are specified within the cryostat tag within the detector definition:

- material
- rmin1, rmin2: the radii of the inner wall, set equal to BarCryoECal_rmin and BarCryoECal_rmin+CryoBarrelFront
- rmax1 and rmax2: the radii of the outer wall, set equal to BarCryoECal_rmax-CryoBarrelBack and BarCryoECal_rmax
- dz: the half length along z, set equal to BarCryoECal_dz

The inner and outer walls and both forward and backward sides of the cryostat can be made passive or active by specifying e.g. <front sensitive="true"/> in the XML. This is used for instance to extract the energy deposited upstream or downstream the sensitive calorimeter in the derivation of dead material corrections.

The bath is modelled as a hollow cylinder (dd4hep::Tube) within the cryostat. The bath properties are specified in the bath tag of the XML file:

- material: the active medium

- `rmin`, `rmax` and `dz`: the dimensions, set equal to `Bath_rmin`, `Bath_rmax`, and `EMBarrel_dz`.

Two `dd4hep::Tube` are built to represent the services between the bath and the cryostat, in the inner and outer radii of the calorimeter.

The calorimeter itself is composed of the electrodes, the absorbers and the gaps. The properties of the calorimeter are defined in the `calorimeter` tag of the XML file:

- the first definitions concern the geometrical dimensions of the cylinder `rmin`, `rmax` and `dz`, equal to `EMBarrel_rmin`, `EMBarrel_rmax`, and `EMBarrel_dz` and an `offset` along ϕ that defines the correspondence between the first absorber plane and its ϕ coordinate.
- Following these definitions are the specifications of the properties of the active gaps, the passive absorbers and the readout electrodes. The `active` tag allows defining the `material` of the active medium as well as the `thickness` (equal to the precomputed `Sensitive_thickness`) of the gaps.
- The `passive` tag contains the properties of the absorbers. It has five sub-tags: `rotation`, `inner`, `innerMax`, `glue` and `outer`. The `rotation` sub-tag allows defining the tilt of the absorbers in $R - \phi$ via its `angle` property (set equal to `InclinationAngle`). The `inner` sub-tag allows specifying the properties of the core of the absorber, such as its `thickness` at the inner radius of the calorimeter (equal to `Pb_thickness`), `material`, and whether it is `sensitive` or not. The `glue` sub-tag allows setting the properties of the glue, i.e. its `thickness` (equal to `Glue_thickness`, `material` and whether it is `sensitive` or not. The `outer` sub-tag defines the properties of the outer part of the absorber: `thickness` (equal to `Steel_thickness`), `material` and whether it is `sensitive` or not. The `innerMax` sub-tag is used to specify the `thickness` (equal to `Pb_thickness_max`) of the absorber at the outer radius of the calorimeter.
- The `readout` tag contains the properties of the electrodes: `thickness` (equal to `readout_thickness`), `material`, and whether it is `sensitive` or not.
- The last tag, `layers`, contains the longitudinal segmentation of the electrode. This is defined by a sequence of instructions like the following: `<layer thickness = "xx*cm" repeat="1"`, where `thickness` is the length of the layer along the electrode direction and `repeat` gives the number of consecutive layers with the same length.

The electrodes, glue and outer sheets of the absorbers are parallelepipeds (`dd4hep::Box`) while the inner part of the absorbers are trapezoids (`dd4hep::Trd1`). The sensitive gaps are `dd4hep::SubtractionSolid` between a `dd4hep::Trd1` representing the volume within two absorbers and the `dd4hep::Box` corresponding to the electrode. Separate Geant4 volumes are created for the sections of the electrodes, absorbers and active medium corresponding to different layers for the study of the per-layer sampling fractions.

The internal consistency between the number of readout planes, gap size, thickness of the absorbers and electrodes and calorimeter radius is checked in the C++ driver: if the numbers are inconsistent, an error message is printed and the simulation is aborted.

In the simulation, the detector readout is implemented via the segmentation class `FCCSWGridModuleThetaMerged_k4geo`. It is specified in the `segmentation` tag within the `readout` field of the XML file defining the calorimeter properties. The class allows the user to group, in each longitudinal layer, an arbitrary number of adjacent argon gaps together along ϕ, and a given number of adjacent cells along θ, using a minimum common granularity. A unique cell identifier is assigned to each cell, the energy of the cell is the sum of the energies in the group, and the position of the cell is at the geometrical centre of the group. The following properties are set within the XML file:

- `offset_theta` (float): minimum value of θ of the digitised cells
- `grid_size_theta` (float): minimum granularity of the readout along the θ direction.
- `mergedCells_Theta` (list of floats): number of adjacent cells along θ to group together in each longitudinal layer.
- `mergedModules` (list of floats): number of adjacent argon gaps to group together in each longitudinal layer
- `nModules` (int): number of electrodes. This quantity is needed to calculate the position of the centre of the cell after multiple gaps are merged together. It is set equal to `ECalBarrelNumPlanes`.

The readout associated to this segmentation class defines the way a unique cell identifier (`CellId`) is assigned to each cell via the `id` XML tag. When a Geant4 step takes place in the LAr gap and energy is transferred to the active medium, a hit is created and added to the output of the simulation, to record information about the particle that interacted, the deposited energy, the hit position and a "Cell ID" bitmap that encodes the system, volume identifier, layer, and index of module and θ physical cell (i.e. after grouping neighbouring cells together). The bitmap used for the barrel has the following format: `system:4,cryo:1,type:3,subtype:3,layer:8,module:11,theta:10`. The meaning of each field is described here-under.

- `system`: 4 bits to encode the sub-detector from which the hit comes from (common to all subsystems)
- `cryo`: 1 bit that defines whether the cell is inside the volume of the cryostat or not (used for dead material correction studies)
- `type`: 3 bits to tell whether the cell belongs to the active medium (0), the absorbers (1) or the readout PCB (2). This field is useful to derive the sampling fractions.
- `subtype`: 3 bits to further tell where in the absorber the hit occurred (cassette, glue or core). This field can be useful for detailed studies on the absorber structure.
- `layer`: 8 bits to identify the longitudinal layer. It starts from zero for the innermost layer to `ECalBarrelNumLayers`-1 in the outermost layer.
- `module`: 11 bits to identify the "ϕ index" of the cell. It starts from zero for the first positioned electrode volume and increases in the anticlockwise direction in the $R-\phi$ plane by one unit for each consecutive electrode. If a given number of adjacent modules are grouped together in a single cell, the cell is assigned the lowest module number of the group (e.g. when grouping by two modules, the module identifier of the cells takes the values 0, 2, 4, ..).

- `theta`: 10 bits to identify the polar index of the cell. It is calculated as (θ - `offset_theta`)/`grid_size_theta`. Similarly to the module number, if a given number of adjacent θ cells are grouped together in a single cell, the cell is assigned the lowest θ index of the group (e.g. when grouping by four cells, the θ identifier of the cells takes the values 0, 4, 8, ..).

4.5.2.2 End-caps

The design of the EM endcap calorimeter is motivated by the following considerations (which are similar to those for the barrel):

- Showers should be sampled frequently, with thin absorbers and noble liquid gaps. Such a structure also allows for highly granular readout.
- The calorimeter should be as uniform in ϕ as possible.
- It should be possible to read out the calorimeter entirely from the downstream (high-$|z|$) faces, to minimise material upstream of the calorimeter.
- The calorimeter should be constructed from multiple copies of just a few types of absorbers and readout boards.
- The calorimeter geometry should allow for a highly granular readout, to enhance π^0/γ separation and facilitate particle-flow reconstruction.

These consideration lead to a "turbine-like" design, where the absorbers and readout boards are arranged as blades. A single unit cell (consisting of an absorber, readout board, and its surrounding noble liquid gaps) is shown in Fig. 27a. This structure is repeated around the full azimuth, resulting in the arrangement shown in Fig. 27b.

One clear drawback to this design is that the noble liquid gap naturally increases with $\rho \equiv \sqrt{x^2 + y^2}$, meaning that the depth (in radiation lengths) and sampling fraction of the detector also vary strongly with ρ. This variation can be mitigated by increasing the thickness of the absorber blades with ρ. However, there are practical limits to the range of ρ over which this approach can be implemented (and also limits on the size of readout boards), which leads to a design consisting of a set of nested 'wheels', where the absorber thickness and noble liquid gap are "reset" at the inner radius of each wheel. This nested-wheel structure is illustrated in Fig. 27c. The structure under investigation currently has three wheels; parametrised calculations of the sampling fraction as a function of radius show that there would be little advantage from having more wheels, and since performance in the transition region from one wheel to the next presents challenges to the calibration, there is a motivation to have as few wheels as possible. The radial extent of each wheel is set such that the ratio of the inner and outer radii is the same for all wheels.

The geometry of the barrel detector is built by the C++ driver `ECalEndcap_Turbine_o1_v03_geo` from the `k4geo` repository [50] based on the properties (geometry, material, readout granularity, ...) specified in XML files in e.g. the `FCCee/ALLEGRO/compact/ALLEGRO_o1_v03` folder.

This concept shares many features with the barrel calorimeter: each end has a cryostat that is modelled as a hollow cylinder (`dd4hep::Tube`), which is filled with a bath of liquefied noble gas. The detector wheels, and their support structures, are

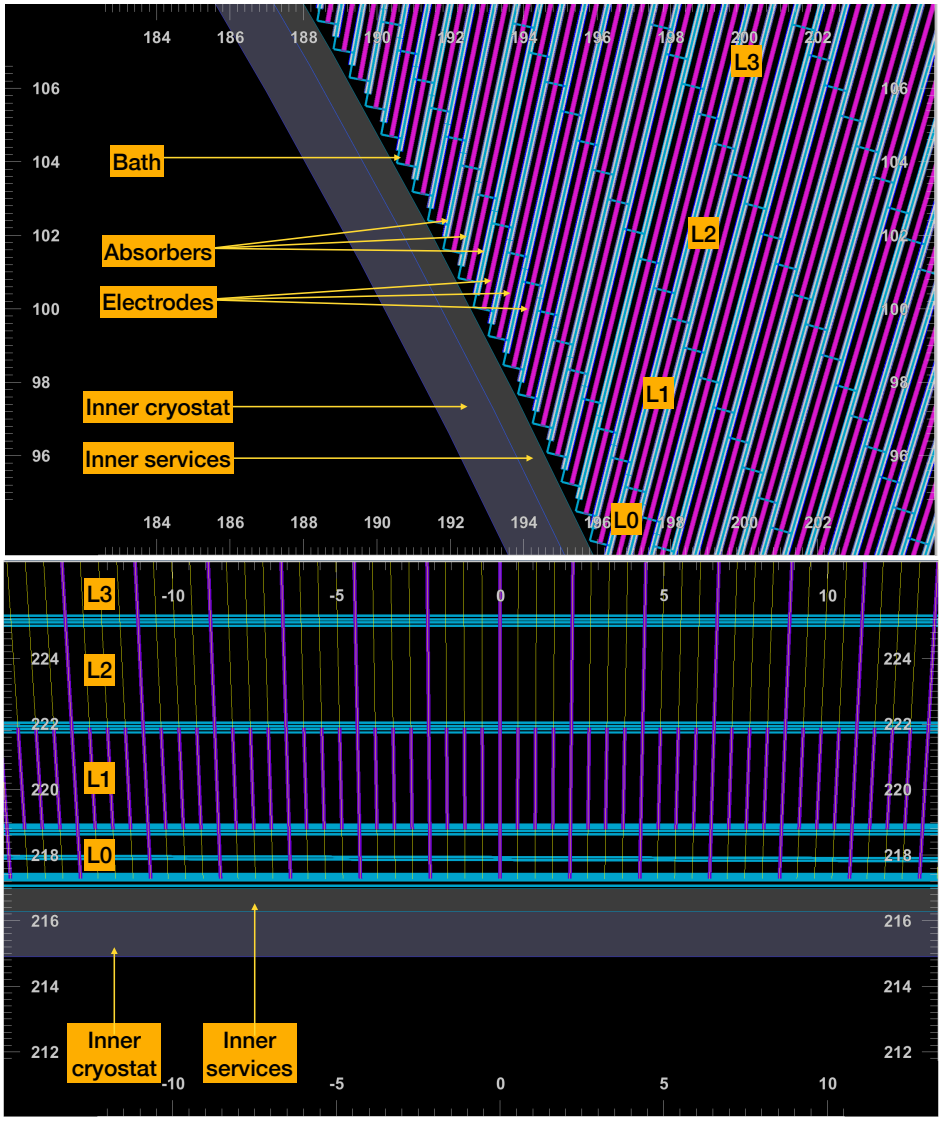
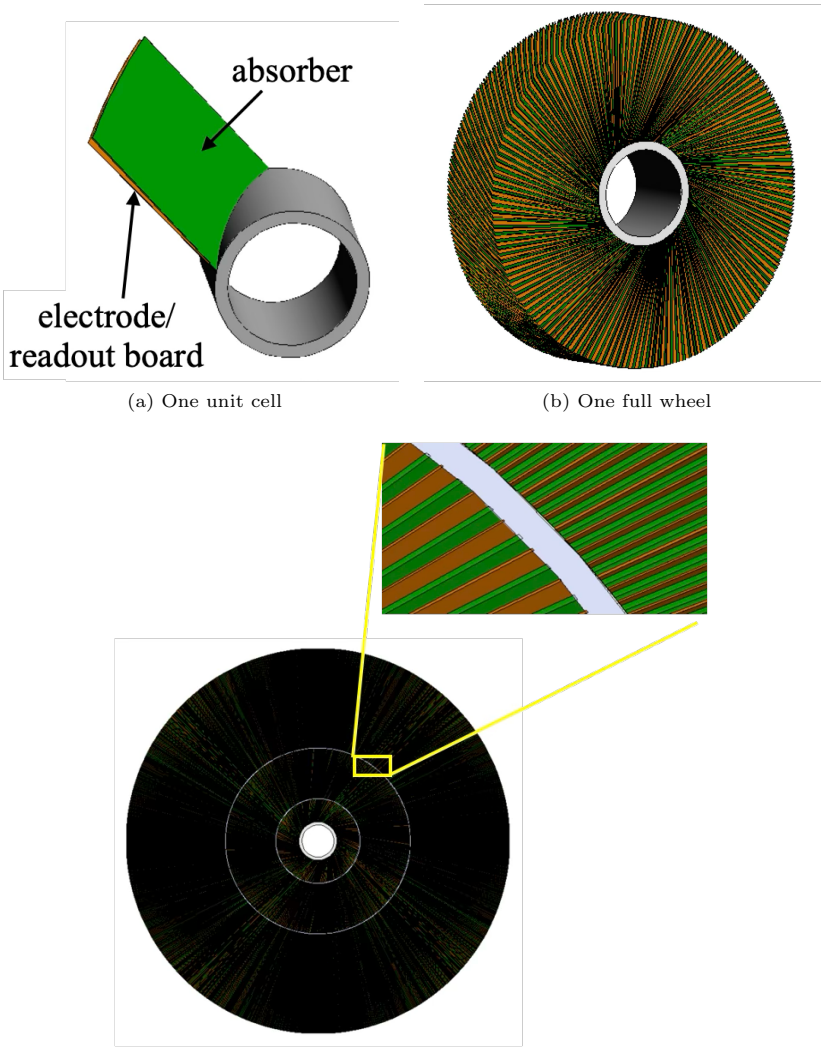


Fig. 26: Geometry implementation in DD4hep of the inner parts of the barrel of the noble liquid electromagnetic calorimeter, as obtained with the `calodisplay` tool [77]: $R-\phi$ view (top) and $R-z$ view (bottom). The inner cryostat, services, and first four layers (L0, L1, L2, L3) are visible. In the $R-\phi$ view the electrodes, absorbers and gaps in between are displayed. In the $R-z$ view, the finest θ grid of θ cells before merging is indicated by the thin yellow lines, while the cell borders after merging are denoted by the thick violet lines. In this example, L1 cells are not merged, while in the other layers the cells are merged in groups of four adjacent cells.



(a) One unit cell

(b) One full wheel

(c) Nested wheels

Fig. 27: Drawings of the "turbine" end-cap geometry: (a) shows a single absorber and readout board, (b) shows a complete set of absorbers and readout boards in ϕ, and (c) shows the nested wheel design, with an inset demonstrating that the noble liquid gaps are relatively large at the outer radius of a wheel, and then "reset" to smaller gaps at the inner radius of the next wheel.

placed within this bath. The support structures are modelled as cylinders that are coaxial with the z axis. Within each wheel, a set of absorbers is placed. Between each pair of absorbers, there is a noble liquid gap, a multi-layer PCB readout board, and another noble liquid gap. One set of absorber/gap/PCB/gap is referred to as a "unit

cell" below. The absorbers and PCBs are placed at an angle with respect to the xy plane. The absorbers, PCBs, and noble liquid gaps are Boolean solids, formed from the intersection of a `dd4hep::Trd2` and a `dd4hep::Tube` covering the active region of the wheel. To model the layers of steel and glue on the outer surfaces of of the absorber, the absorber is taken to consist of three nested volumes: the outer (thickest) volume is steel, the next volume is glue, and the final (thinnest) volume is the absorber material. The endcap at the $-z$ side of the detector is constructed by mirroring the $+z$ endcap.

Within the turbine concept, there are several parameters that can be varied: the angle of the blades with respect to the xy plane, the thickness of the absorbers at the inner radius of each wheel, the scaling of the absorber thickness with ρ, the thickness of the glue and steel layers that clad the absorbers, and the number of absorber/gap/-electrode/gap unit cells in ϕ (setting this value is equivalent to, but more convenient than, setting the noble liquid gap width). These are set by the parameters described below.

Parameters defined in `DectDimensions.xml`, which sets the overall sizes of the ALLEGRO subsystems:

- `ECalEndcap_inner_radius`: inner radius of the calorimeter volume.
- `ECalEndcap_outer_radius`: outer radius of the calorimeter volume.
- `ECalEndcap_min_z`: position of the inner $|z|$ face of the calorimeter volume.
- `ECalEndcap_max_z`: position of the outer $|z|$ face of the calorimeter volume.

Parameters set in the subsystem-specific XML file `ECalEndcaps_Turbine_o1_v03.xml` :

- `CryoEMECThicknessFront`: thickness of the front wall of the cryostat.
- `CryoEMECThicknessBack`: thickness of the back wall of the cryostat.
- `CryoEMECThicknessInner`: thickness of the inner-radius wall of the cryostat.
- `CryoEMECThicknessOuter`: thickness of the outer-radius wall of the cryostat.
- `CryoEndcap_front_rmin`: inner radius of the endcap cryostat, at the inner face in $|z|$. Typically set equal to `ECalEndcap_inner_radius`.
- `CryoEndcap_back_rmin`: inner radius of the endcap cryostat, at the outer face in $|z|$. Note that in the design currently under study, the cryostat is taken to be cylindrical, so that `CryoEndcap_front_rmin` = `CryoEndcap_back_rmin`, both being set equal to `ECalEndcap_inner_radius`.
- `CryoEndcap_rmax`: outer radius of the endcap cryostat. Typically set equal to `ECalEndcap_outer_radius`.
- `CryoEndcap_z1`: the inner extent of the endcap cryostat in $|z|$. Typically set equal to `ECalEndcap_min_z`.
- `CryoEndcap_z2`: the outer extent of the endcap cryostat in $|z|$. Typically set equal to `ECalEndcap_max_z`.
- `BathThicknessFront`: gap between the cryostat wall and the detector on the low-$|z|$ side.
- `BathThicknessBack`: gap between the cryostat wall and the detector on the high-$|z|$ side.
- `BathThicknessOuter`: gap between the cryostat wall and the detector at the outer radius.

- `nWheels`: number of wheels in the detector.
- `ElectrodeBladeThickness`: thickness of the readout electrodes.
- `BladeAngle[N]`: angle of the blades with respect to the xy plane in the Nth wheel of the detector.
- `AbsorberBladeThickness[N]`: thickness of the absorbers at the inner radius of the Nth wheel. Note that this thickness includes the glue and steel plating on the faces of the absorbers.
- `AbsorberBladeThicknessScalefactor[N]`: the rate at which the thickness of the absorbers increases with increasing ρ for the Nth wheel. This is the parameter f in the equation below:
$$t_A(\rho) = t_A(\rho_i)\left(1 + f\left[\frac{\rho - \rho_i}{\rho_i}\right]\right)$$
where $t_A(\rho)$ is the absorber thickness at ρ, ρ_i is the inner radius of the wheel, and $t_A(\rho_i)$ is the absorber thickness at the inner radius.
- `nUnitCells[N]`: the number of absorber/gap/electrode/gap until cells in wheel N.
- `ECalEndcapSupportTubeThickness`: thickness of the mechanical support at the interface between two wheels.
- `EMEC_steel_thickness` thickness of the steel plating on the absorber surfaces.
- `EMEC_glue_thickness` thickness of the glue that adheres the steel plating to the absorber surfaces.

In the turbine geometry, the noble liquid gap and the sampling fraction can vary as a function of both ρ and z. Therefore the noble liquid volume is subdivided into cells in both dimensions, allowing for separate calibration constants to be derived for cell having different $z \times \rho$ indices (to keep the terminology similar to that used in the barrel calorimeter, each group of cells sharing the same calibration constant is called a 'layer'). The arrangement of the calibration cells is shown in Fig. 28. The calibration granularity is determined by the following parameters in the XML:

- `ECalEndcapNumCalibRhoLayersWheel[N]`: number of calibration cells in the ρ direction for wheel N.
- `ECalEndcapNumCalibZLayersWheel[N]`: number of calibration cells in the z direction for wheel N.

Given its unique geometry, a custom readout is used for the turbine calorimeter. The readout cells are arranged with the same ρ/z segmentation as is used for the calibration cells, though the readout cells may be smaller. The number of readout cells in each dimension is required to be an integer multiple of the number of calibration cells in that dimension, so that a given readout cell never spans calibration cells; this requirement is enforced in the detector-building code. This readout is implemented via the segmentation class `FCCSWEndcapTurbine_k4geo`. It is specified in the segmentation tag within the readout field of the XML file defining the calorimeter properties. The following properties are set within the XML file:

- `ECalEndcapNumReadoutRhoLayersWheel[N]`: number of readout cells in the ρ direction for wheel N.
- `ECalEndcapNumReadoutZLayersWheel[N]`: number of readout cells in the z direction for wheel N.

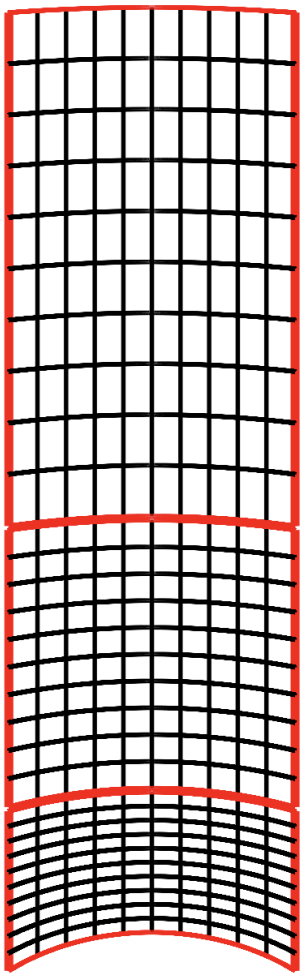


Fig. 28: Schematic drawing of the calibration or readout cells for a single PCB in all three wheels of the endcap EM calormeter, as seen in a face-one view of the PCB (i.e. the board lies in the plane of the page). The index z is used to number the vertical strips, and the index ρ is used to number the horizontal arcs.

- `offset_theta` (float): minimum value of θ of the digitised cells. This is needed for the sliding windows cluster reconstruction algorithm.
- `offset_z` (vector of floats): z position of the centre of the $+z$ endcap. The center of the $-z$ endcap is assumed to be at $-$`offset_z`. The vector contains one value per wheel (though in the nominal design the value is the same for all wheels), and the values are calculated automatically from the user-provided parameters.

- `grid_size_z` (vector of floats): extent of a readout cell in z. The vector contains one value per wheel, and the values are calculated automatically from the user-provided parameters.
- `grid_size_rho` (vector of floats): extent of a readout cell in ρ. The vector contains one value per wheel, and the values are calculated automatically from the user-provided parameters.
- `mergedModules` (vector of ints): number of unit cells to merge (effectively summing a set of readout cells in ϕ). The vector contains one value per wheel.

The bitmap associated with the readout has the following format: `system:4,cryo:1,type:3,subtype:3,side:-2,wheel:3,layer:12,module:11,rho:8,z:8` where

- `system`: 4 bits to encode the sub-detector from which the hit arises (common to all subsystems)
- `cryo`: 1 bit that defines whether a hit is inside the volume of the cryostat or not (useful for dead material correction studies)
- `type`: 3 bits that are set according to whether the hit occurs in the active medium (value = 0), the absorbers (1), or the readout PCB (2). Can be used e.g. to derive the sampling fractions.
- `subtype`: 3 bits to further refine the location of hits in the absorber. A value of 0 means the hit was in the core of the absorber, 1 means the hit was in the glue layer, and 2 means the hit was in the steel cladding.
- `side`: 2 signed bits to mark whether the hit was in the $+z$ or $-z$ endcap.
- `wheel`: 3 bits to determine which detector wheel the hit occurred in. Wheels are numbered starting at 0 for the innermost in ρ.
- `layer`: 12 bits to determine the calibration layer for the hit. Since the sampling fraction is expected to depend on both ρ and z, calibration is done in a two-dimensional grid on the face of the readout board, and the layer index specifies a cell in that grid.
- `module`: 11 bits to identity the readout PCB where the hit was recorded.
- `rho`: 8 bits to denote the ρ position of the cell relative to `offset_rho`.
- `z`: 8 bits to denote the z position of the cell relative to $\pm$`offset_z`.

4.5.3 Segmented crystal EM precision calorimeter (SCEPCal)

The SCEPCal is a quasi-projective geometry of longitudinally segmented homogeneous crystals with dual readout capabilities providing excellent EM resolution. Angular offsets from the interaction point are used to mitigate projective cracks. A pure-projective precision crystal timing layer is instrumented in front of the main segmented layer. The following description is adapted in part from [78].

4.5.3.1 Geometry

The detector geometry construction is fully parametrised and automated with the provision of the input parameters listed in Table 3, which are then used to derive the secondary parameters listed in Table 4. The inner radius, z-extent of the barrel, and number of total segments in phi set the overall scale of the detector. A single global

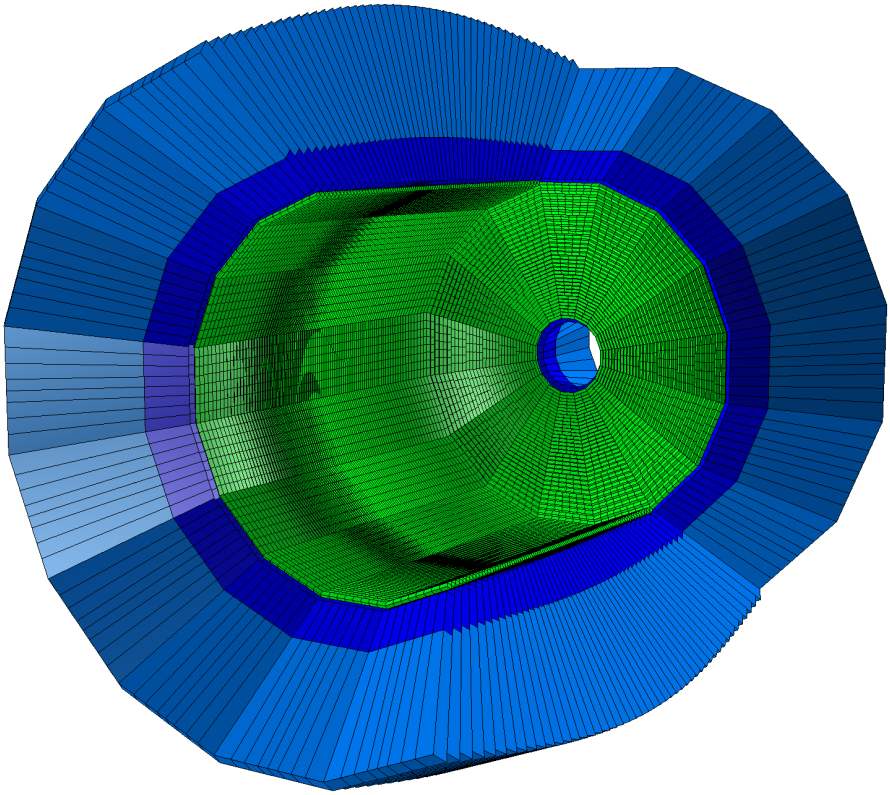


Fig. 29: The Segmented Crystal EM Precision Calorimeter (SCEPCal). Crystal dimensions exaggerated for visibility. Dark/light blue: main layer front/rear quasi-projective crystals. Green: projective timing layer crystals.

phi segment is further segmented in theta and phi into blocks of crystal towers, within which different granularities of front and rear crystals are possible. The nominal values of the square crystal tower face-widths and number of front and rear divisions are used to calculate the final dimensions. Final crystal counts and other outputs are shown in Table 5.

The top level loop of the construction routine loops over the number of global phi segments. A single global phi slice (Fig. 30) is then divided in theta into envelope volumes of asymmetrical trapezoids of `dd4hep::EightPointSolid` according to the nominal crystal tower face width along the z-axis. Crystal towers are then formed by dividing the xy-extent of the chord formed by the global phi slice by the nominal tower face width to form quasi-square towers. Each tower is then split into front and rear crystals, also asymmetrical trapezoids, by the number of respective divisions. The rear vertices of the individual crystal trapezoids are additionally shifted by the ratio of the projective offset to the radial distance of the crystal from the IP to effectively point the crystal away from the IP by the offset distance, mitigating projective cracks.

Table 3: Input parameters for the SCEPCal fully parametrised geometry construction routine. Table from [78].

Description	Variable	Value
Half z-extent of the barrel	Z_{B}	2.40 m
Inner radius of the barrel	R_{inner}	2.25 m
Global number of phi segments	N_{Φ}	128
Nominal square face width of a crystal tower	C_{fw}	10 mm
Number of front crystal divisions per tower	N_{F}	1
Number of rear crystal divisions per tower	N_{R}	1
Front crystal length	F_{dz}	50 mm
Rear crystal length	R_{dz}	150 mm
Pointing offset from IP	P_{offset}	100 mm

Table 4: Secondary parameters for the SCEPCal calculated from input parameters. Table from [78].

Description	Variable	Formula
Angular size of a single phi segment	$d\Phi$	$2\pi/\ N_{\Phi}$
Angular size of barrel region	Θ_B	$\mathrm{atan}(Z_{\mathrm{B}}/R_{\mathrm{inner}})$
Angular size of endcap region	Θ_E	$\mathrm{atan}(R_{\mathrm{inner}}/Z_{\mathrm{B}})$
Number of barrel towers in θ	$N\theta_B$	$\mathrm{floor}(2\ Z_{\mathrm{B}}/C_{\mathrm{fw}})$
Number of endcap towers in θ	$N\theta_E$	$\mathrm{floor}(R_{\mathrm{inner}}/C_{\mathrm{fw}})$
Angular size of a single barrel tower in θ	$d\theta_B$	$(\pi - 2\ \Theta_E)/N\theta_B$
Angular size of a single endcap tower in θ	$d\theta_E$	$\Theta_E/N\theta_E$
Number of barrel segments in ϕ in a single phi segment	$N\phi_B$	$\mathrm{floor}(2\pi R_{\mathrm{inner}}/(N_{\Phi}C_{\mathrm{fw}}))$
Number of endcap segments in ϕ in a single phi segment*	$N\phi_E^*$	$\mathrm{floor}(2\pi R^*_{\mathrm{inner}}/(N_{\Phi}C_{\mathrm{fw}}))$
Angular size of barrel segments in ϕ	$d\phi_B$	$d\Phi/N\phi_B$
Angular size of endcap segments in ϕ	$d\phi_E^*$	$d\Phi/N\phi_E^*$
$*$ varies with R_{inner}		

Table 5: Final crystal/readout counts and endcap dimensions from the parametrised geometry construction using input values from Table 3.

Quantity	Value
Number of total barrel crystals	1,360,128
Number of barrel front crystal readout channels (1 SiPM per crystal)	680,064
Number of barrel rear crystal readout channels (2 SiPMs per crystal)	1,360,128
Number of total endcap crystals	251,136
Number of endcap front crystal readout channels (1 SiPM per crystal)	125,568
Number of endcap rear crystal readout channels (2 SiPMs per crystal)	251,136
Endcap innermost radius (centre point of innermost front crystal face)	360.2 mm
Endcap outermost radius (centre point of outermost rear crystal face)	2386.8 mm
Endcap maximum pseudo-rapidity	2.595

Each crystal tower captures a constant solid angle and is therefore unique in the global phi slice. The basic assumptions for the calculation are that the centre points of the tower front faces are collinear in z along the inner radius of the barrel. The side lengths of the trapezoid faces are calculated by assuming the front face is normal to the

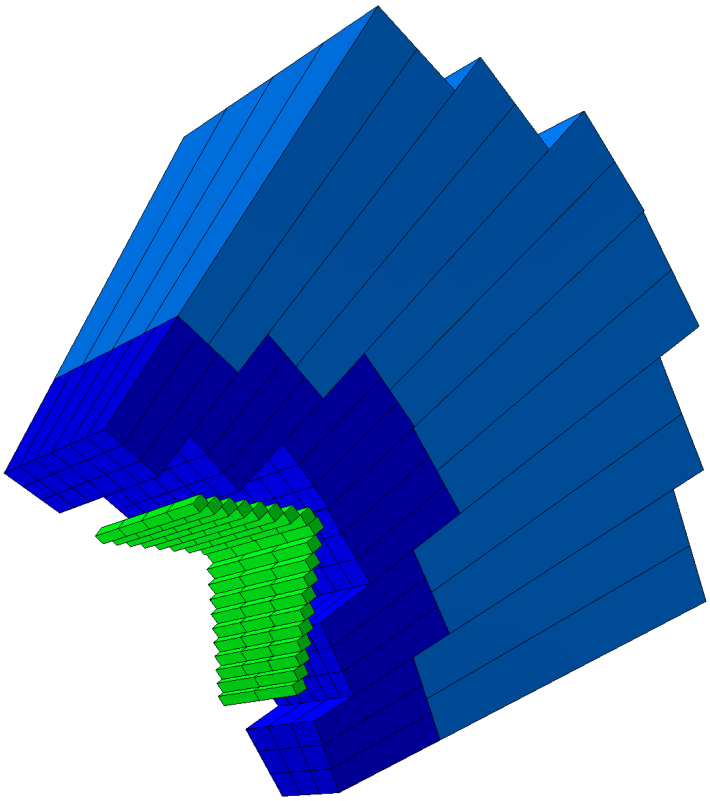


Fig. 30: Barrel/endcap junction of a single global phi slice showing a 3F/2R division in the main layer. Crystal dimensions exaggerated for visibility. Dark/light blue: main layer front/rear quasi-projective crystals. Green: projective timing layer crystals.

IP, with an inner radius equal to that provided plus the projective offset, resulting in the crystals pointing "through" the IP to mitigate projective cracks. The side lengths are then shrunk by the appropriate factor when placing the crystals along the true inner radius. The crystal lengths are fixed by the given inputs, resulting in the familiar teeth-like pattern of projective ECALs such as CMS and L3.

Assembly volumes (`dd4hep::Polyhedra`) for the main and timing layers are made separately to avoid overlaps with the other sub-detectors, matching the z-extent, inner radius, and phi segmentation provided. The front, rear, and side protrusions due the tilted projective geometry are calculated to snugly fit all crystal towers in the polyhedra volumes. The construction routine in the endcap is similar as in the barrel, but the inner radius is updated for each theta.

The timing layer is a purely projective single layer of crystals. The nominal depth and length are used to calculate integer numbers of crystals for the array.

Because a projective implementation is not already built into DD4hep, the projective calculations are calculated specifically for this detector. The global coordinates for each crystal centre, as mapping from the `cellID`, are saved into the segmentation for later recall in the sensitive action to record the hit positions.

With the parameters in Table 3, the full geometry is built in under a minute with a memory footprint around 8 GB. The SCEPCal simulation code is available at [79] and is being integrated into k4geo. The constructors are `SCEPCal_MainLayer.cpp` and `SCEPCal_TimingLayer.cpp`.

4.5.3.2 Outlook

The current implementation is an idealised geometry with no spacing for instrumentation or cooling elements. Gaps to accommodate these additional elements can be added easily in view of the generalised calculations. The main layer and timing layer are implemented as separate constructors, but share some parameters defining their relative placements. Care should be taken to ensure these dimensions are always consistent. A custom sensitive action is implemented to handle the dual readout capabilities of the detector, recording scintillation and Cerenkov photon counts in separate readout collections in place of the energy deposit.

4.5.4 Monolithic fibre dual readout calorimeter

A hadronic particle that developed destructive interaction within a calorimeter generates not only hadronic components but also electromagnetic (EM) components due to the decay of neutral pions into photons. The primary challenge in measuring the energy of hadronic particles lies in the fact that the response to the hadronic component is significantly lower than that of the EM component for most active materials. Since the fraction of EM components within a hadronic shower broadly fluctuates event by event, the difference in response to each component leads to poor hadronic energy resolution compared to the EM energy resolution.

The Dual-Readout calorimeter is a proposed solution to improve the hadronic energy resolution by introducing two independent readout channels, Cherenkov and scintillation, with distinctive response ratios for the hadronic to EM component (h/e) [80]. The Cherenkov channel is mainly sensitive to light and charged particles such as electrons by the nature of Cherenkov radiation, having a lower h/e response than the scintillation channel. Therefore, the utilisation of two readout channels allows the measurement of the fraction of EM components in a hadronic shower on an event-by-event basis, offering excellent hadronic energy resolution.

The "monolithic" Dual-Readout calorimeter is the baseline design considered since the FCC-ee CDR [41], without longitudinal segmentation or geometry adaptation. The calorimeter is segmented into numerous towers made of a cuboid-shaped copper absorber with trapezoid front and rear faces. The edges connecting the front and rear faces are aligned to the interaction point of the detector, forming a projective geometry. The distance between the front and rear face is 2.0 m. The scintillation and Cherenkov optical fibres are inserted in a checkerboard pattern, where the fibres are perpendicular to the front and rear faces to ensure a uniform sampling fraction regardless of the

shower depth. Each fibre has a 1.0 mm diameter, and the distance between the cores of neighbouring fibres is 1.5 mm. Because of the projective geometry, the fibres inserted in the wing part of the trapezoid at the top face may have lengths shorter than 2.0 m. The length of a fibre is truncated to the point where the fibre meets the tower edge.

Both barrel and endcap calorimeters have rotational symmetry in ϕ-direction, segmented into 144 slices. Each slice in the barrel consists of 40 towers covering the range $0 < |\theta - \pi/2| < \pi/4$, while each slice in the endcap holds 35 towers covering the range $\pi/4 < |\theta - \pi/2| < (15/32)\pi$. Every tower has the same opening angle of $\pi/160$ with respect to the interaction point. The radius between the beam axis and the front faces of towers in the barrel is 2.5 m, and the distance between the interaction point and the front faces of towers in the endcap is also 2.5 m. Figure 31 shows a visualisation of the single tower and the full geometry of the Dual-Readout calorimeter.

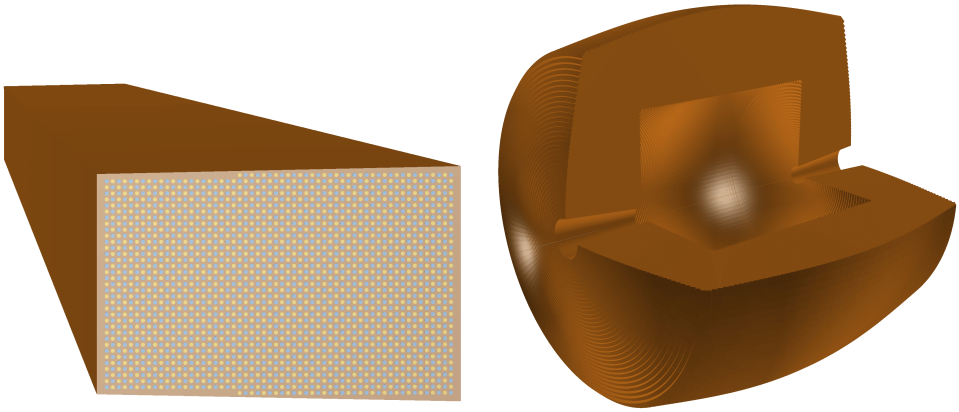


Fig. 31: A drawing of (left) the single tower seen from the rear side and (right) the full geometry of the Dual-Readout calorimeter. Because of the projective geometry, fibres are placed only if there are enough spaces to insert the fibre perpendicularly into the tower.

The material properties of scintillation and Cherenkov fibres follow the description of Kuraray SCSF-78 [81–83] and Mitsubishi Eska SK40 [84, 85] optical fibres. The high light yield of a scintillation fibre is expected to stress the dynamic range of a silicon photomultiplier (SiPM). Also, the ultraviolet (UV) component has a relatively shorter attenuation length, which can cause the light yield dependency as a function of the shower depth. Therefore, Kodak Wratten number 9 yellow filter [86] is planned to be attached to the scintillation channel, and its filtering sequence is described in the simulation.

The geometry of the monolithic Dual-Readout calorimeter is governed by the `DDSegmentation` class `GridDRcalo_k4geo`. The class owns two instances named `DRparamBarrel_k4geo` and `DRparamEndcap_k4geo`, where both of them are inherited from the base class `DRparamBase_k4geo`. `DRparamBarrel_k4geo` and `DRparamEndcap_k4geo` calculate four 3D vectors per tower where each vector points

to the centre of the edge that connects side faces in $+\phi$ and $-\phi$ directions. The base class `DRparamBase_k4geo` describes methods to calculate the necessary dimensions to construct `dd4hep::Trap` corresponds to each tower.

The implementation of DD4hep volumes is performed inside the class `ddDRcalo::DRconstructor`. For each tower in the η-direction, the `dd4hep::Volume` instance is created within the function `ddDRcalo::DRconstructor::implementTowers` based on the `dd4hep::Trap` above. Then, the tower is replicated in the ϕ-direction to construct the 4π geometry. The placements of fibres with respect to the tower are calculated inside the method `ddDRcalo::DRconstructor::implementFibers`. To reduce memory consumption, a unit of the checkerboard pattern called `unitBox` is formed, which consists of two Cherenkov and two scintillation fibres. The `unitBox` is replicated in places where the fibre length equals the tower height. In the wing part of the trapezoid tower, fibres are cropped to the points where the edge meets the side face of the tower using the function `ddDRcalo::DRconstructor::calculateFiberLen` to prevent the extrusion of fibres. Finally, the core and cladding of each fibre are constructed within the method `ddDRcalo::DRconstructor::implementFiber`.

4.5.5 Capillary tubes fibre dual readout calorimeter

The capillary tube dual readout calorimeter is a geometry variation of the 'Monolithic Fibre Dual Readout Calorimeter' described in Section 4.5.4. In this geometry each fibre is placed in a metallic tube which acts as absorber material for the calorimeter. The tubes are stacked together in a hexagonal pattern to form towers, which follow a similar general shape as the monolithic fibre calorimeter.

The implementation of this geometry was recently carried out and both the projective calorimeter barrel and endcap regions are available as DD4hep sub-detectors. An image of the barrel and endcap geometries can be seen in Fig. 32 and Fig. 33.

4.5.5.1 Implementation details

The barrel and endcap geometries are implemented as two separate sub-detectors, which can be simulated independently from one another, allowing their parameters to be tuned separately. In the baseline design, they are assumed to touch at an angle of 45°. The parameter sets describing the barrel and endcap geometries are listed in Table 6 and Table 7.

From these sets of parameters, the full geometries are created. In the barrel region, the tower length is calculated from the difference between inner and outer radius, with a strict requirement of `InnerCaloRadius`<`OuterCaloRadius`. `InnerCaloHalfLength` is the distance from the interaction point to the front face of the calorimeter in the z-direction. Under the assumption of meeting the endcap at 45°, this value needs to be the same as `InnerCaloRadius`.

Each endcap is designed to form a 90° angle with the barrel. Similarly to the barrel, the free parameters are the inner radius and the tower length, which then defines the outer radius.

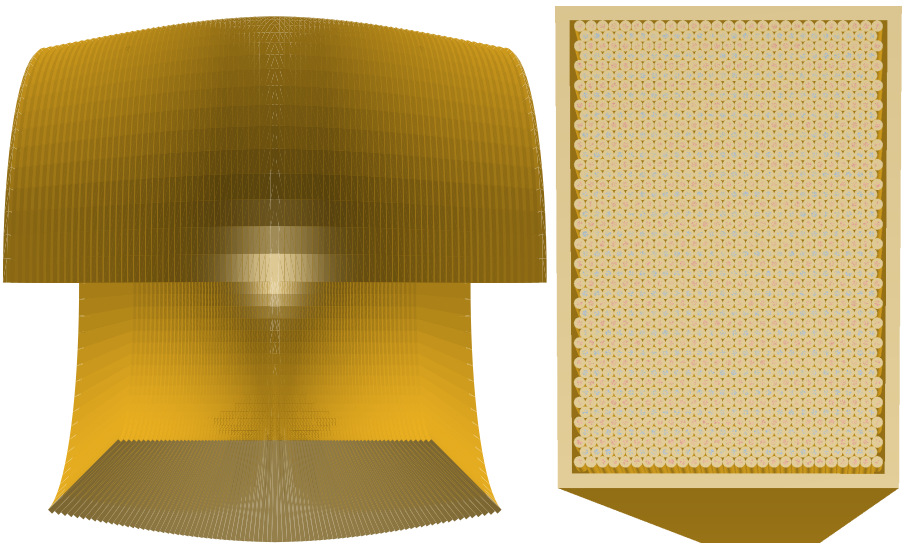


Fig. 32: Barrel region of the dual readout calorimeter (left). Backside of one tower with the assembly of tubes visible, forming one tower (right).

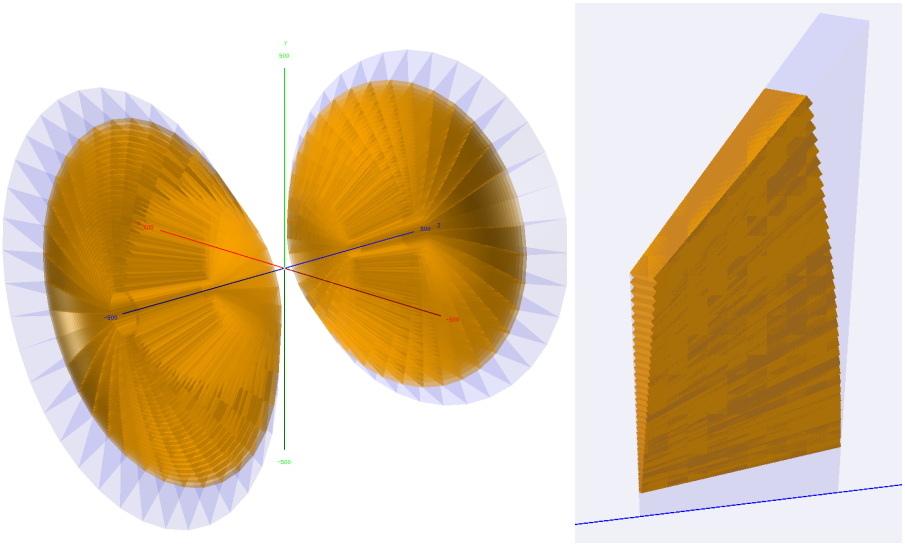


Fig. 33: Endcap region of the dual readout calorimeter, considering 35 θ-towers and 36-rotations around the beam pipe. The trapezoidal structure (a ϕ-slice) containing the θ-towers is displayed in light blue (left). A single ϕ-slice populated with 35 θ-towers (right).

Due to the rotational symmetry in ϕ, the barrel and endcaps are both created by multiple placings of one *stave*, which is the collection of towers at a fixed ϕ (i.e. one slice). The stave volume is a *dd4hep::Trap* (see Fig. 33), and the collection of staves

Name	Default Value
DRBTInnerCaloRadius	2.8 m
DRBTOuterCaloRadius	4.6 m
DRBTInnerCaloHalfLength	2.8 m
DRBTTowerThetaCoverage	1°
DRBTTowerPhiCoverage	5°
DRBTSupportSideThickness	1 mm
DRBTSupportFrontThickness	1 mm
DRBTSupportBackThickness	0 mm
DRBTTubeOuterRadius	1 mm
DRBTCladOuterRadius	0.5 mm
DRBTCOreOuterRadius	0.485 mm
DRBTTubeTolerance	50 µm

Table 6: Barrel

Name	Default Value
DRETinnerRadius	2.8 m
DRETtowerHeight	1.8 m
DRETNbOfZRot	72
DRETTubeRadius	1 mm
DRETCladRadius	0.5 mm
DRETCoreRadius	0.45 mm

Table 7: Endcap

Table 8: Set of parameters describing the (a) barrel and (b) endcap geometries of the capillary tube dual readout calorimeter.

is an *Assembly* volume. For both barrel and endcaps, this forms the highest hierarchy volume in the sub-detector's geometry.

To specify the lateral size of the barrel towers, the angular coverage of one tower in θ and ϕ can be set with the corresponding parameters. In the endcaps, the ϕ size is set by the number of rotations about the z axis, with the default value of 72 corresponding to 5°. When simulating endcaps and barrel together, the ϕ coverage needs to be identical to avoid overlaps. For the endcaps, the θ coverage is fixed to 1.125° at the moment.

The stave is then populated with towers arranged in θ, where each tower is different, apart from a forward-backward symmetry. The bounding box of a tower is a *Trap* volume, which in the barrel region also acts as a support structure with adjustable thickness and material (the default material is the same as the absorber tubes, see below). It is "hollowed out" by a daughter volume of air into which the tubes are placed. For the endcaps, the shrinking size of towers close to the beam pipe prevents this method of implementing the support. Therefore, this 'support' volume is skipped and the tower directly consists of the air *Trap*. The dimensions for the tower are all calculated from the parameters named thus far. The implementation will throw an error if the tower is too small to host a single tube. This tower *Trap* is the next volume in the hierarchy after the stave, and is the first volume to correspond to a physical object in the detector.

All the tubes of the tower are placed inside the air volume of the tower. The radius of the tubes can be tuned with the corresponding tube radius parameters. The tubes are realised in the simulation as a *Tube* volume with inner radius of 0 mm and the set outer radius. The length is determined by the position in the tower due to the trapezoidal shape, where in the barrel region, the length is rounded down to the next multiple of the `DRBTTubeTolerance` parameter. The material can be set in the XML file, in the current implementation it is set to be brass. Since there is no subdivision of

the tower due to the complicated geometry, the highest number of daughter volumes in the geometry is likely to occur here.

To create the active calorimeter medium, the tube volume is filled with the material forming the optical fibres. The daughter volume to the tube is the cladding of the fibre. The radius is determined by the cladding radius parameter, the inner radius is set to 0 mm, the length is the same as the tube's length which is hosting the fibre.

Inside the cladding volume, the fibre core is placed, granddaughter of the tube volume. It is the final volume in the hierarchy. There is a protection to ensure *tube radius*>*clad radius*>*core radius*.

All the parameters in the XML file can be changed independently, and the geometry will adapt accordingly. As long as they are kept in a sensible range, no malfunction is to be expected.

The `volID` largely reflects the volume hierarchy for the calorimeter. Inside the tower, the tubes are categorised into rows and columns, following the offset coordinate system for hexagonal planes.[2]

4.5.5.2 Further comments

Both the barrel and endcap geometries are completed and available as DD4hep sub detectors. They are included in a new IDEA detector concept, called IDEA *option 2*. They have been tested successfully for overlaps and memory footprint. In addition, a custom sensitive detector action has been developed for such detectors which allows to simulate 10 GeV electron events at a rate between 0.7 and 1.0 seconds per event, with a small dependence over the showering position in the detector geometry. The event rate scales linearly with the primary particle energy. For more details, see Section 5.2.2.

Due to the large number of tubes in the towers, the capillary dual readout calorimeter is to be used with the new `regexSensitiveDetector`, for matching the custom sensitive action to the tube volumes.

4.5.6 Scintillating tile calorimeter

The hadronic tile calorimeter is a sampling calorimeter made of stainless steel and scintillating plastic tiles, which emit light when traversed by charged particles. The light produced by scintillating tiles is captured and transported via wavelength-shifting fibres to the silicon photomultipliers located at the outer radius of the calorimeter.

The calorimeter is structured into two central barrels and two endcap sections. Each section is further divided into 128 modules in the azimuthal direction and the number of readout modules in ϕ can be configured via an XML file to accommodate specific experimental needs. The orientation of the scintillating tiles perpendicular to the beam line, in combination with wavelength-shifting fibre readout, allows for almost hermetic azimuthal calorimeter coverage.

The barrel calorimeters are composed of one cylinder with fixed inner radius size along the Z axis and fixed number of radial layers. Each endcap calorimeter consists of three cylinders build along the Z axis, each with a different inner radius and number/dimensions of radial layers as indicated in Table 9. The inner radius of each

[2] https://www.redblobgames.com/grids/hexagons/#coordinates-offset

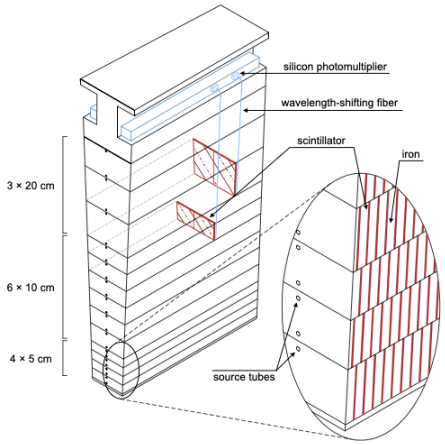


Fig. 34: Schematic view of one module of the hadronic barrel tile calorimeter. The scintillating tiles are shown in red while the wavelength shifting fibre and the silicon photomultipliers are shown in blue.

cylinder, as well as the number and dimensions of radial layers inside each cylinder, can be adjusted in the XML file and these are then used to calculate the cell positions. However, to increase the number of cylinders of endcap calorimeter, one would need to modify the C++ builder of the detector.

The geometry of the barrel module is illustrated in Fig. 34. Each barrel and endcap module contains 2 scintillating tiles per radial layer, which will be separated in ϕ via a reflective material and read out by wavelength shifting (WLS) fibres into two separate silicon photomultipliers (SiPMs).

Each radial layer represents an individual detector volume that hosts the volumes of individual unit periods (sequences) within each layer. These unit periods are individually placed in each radial layer to fill the detector volume along the Z axis. In the current implementation, each unit period of the calorimeter, measuring 18 mm, consists of the following items:

- Two master plates (5 mm each, stainless steel)
- One spacer plate (4 mm, stainless steel)
- One scintillator tile (3 mm, polystyrene)

Table 9: Dimensions of the scintillating tile calorimeter barrel and endcap calorimeters.

	inner radius [cm]	outer radius [cm]	Z min [cm]	Z max [cm]	# of radial layers
Barrel	280	450	0	280	13
Endcap	360	450	290	340	6
	290		340	390	9
	35		390	545	22

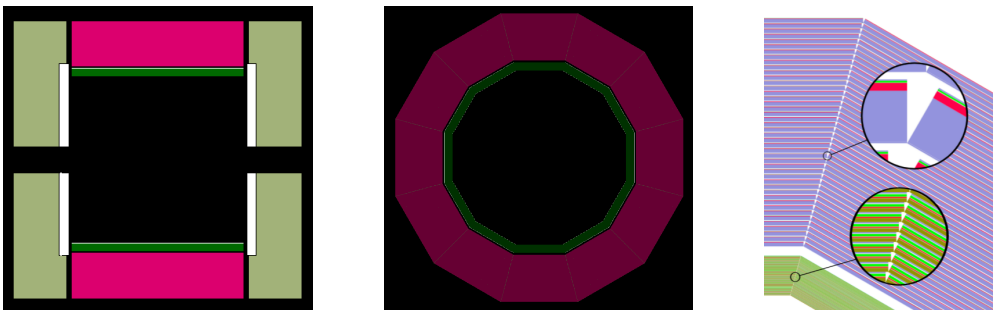


Fig. 35: The left image shows a sagittal view, and the center image presents a transverse cut of the ECAL and HCAL of the CLD detector. The transverse cut is taken at the centre, revealing the 12-sided barrel structure. The image on the right replicates Fig. 20 from [54], depicting a transverse cut in the barrel region and highlighting the gaps between layers at the same radius.

- Two air gaps (0.5 mm each)

The thicknesses of the absorber and scintillator plates can also be defined in this configuration file. Altering the dimensions of the absorber plates affects the number of unit periods that can fit within a given detector dimension. The detector employs two alternating sequences (A and B) to structure the radial layers, with each layer exclusively using one sequence type. The difference between Sequence A and Sequence B lies in the arrangement of the components within each period. The placement of different materials within each sequence can be modified in the XML file. This alternating use of sequences (A in one layer, B in the next) ensures a uniform distribution of materials and an efficient filling of the detector volume in the radial direction. Sequence details:

- sequence A: master - spacer - master - air - scintillator - air
- sequence B: master - air - scintillator - air - master - spacer

4.6 Generic detector builders

4.6.1 Layered polyhedron

The drivers `GenericCalBarrel_o1_v01` and `GenericCalEndcap_o1_v01` are used to create barrel or disk detector shape by piling up layers radially or along the z-direction respectively. Each layer repeats the same sequence of materials and makes up the sides or bases of a polyhedron, for the barrel and end-cap respectively. Due to their rectangular shape, the layers leave gaps along the edges of the barrel.

The main parameters provided to the detector driver in the compact file include the inner and outer geometry, the number of layers, and their composition. The readout granularity is determined by the corresponding DD4hep virtual segmentation. These drivers are versatile and are currently used to describe the ECAL, HCAL, and muon system sub-detectors of the CLD detector, as shown in Fig. 35.

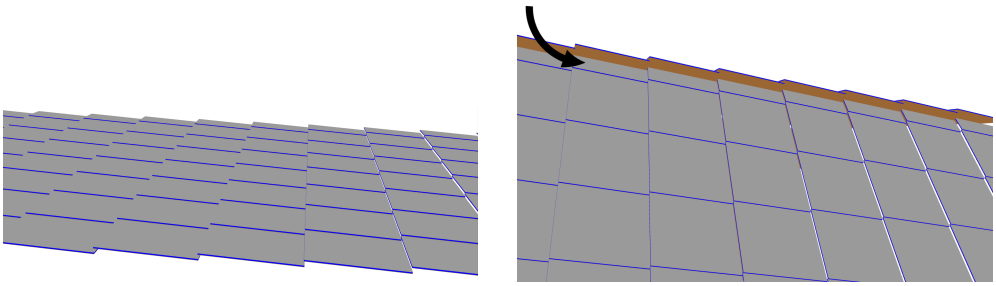


Fig. 36: Left: Two-dimensional overlap between the chambers in one polyhedron side. Right: A chamber with non-standard dimensions at the end of a side.

4.6.2 Hermetic layered polyhedron

The purpose of this driver is to construct a fully polyhedral-shaped sub-detector (Barrel and Endcap) composed of multiple layers of boxes, which overlap to eliminate cracks in the sensitive regions. The overlap occurs in two dimensions, as shown in Fig. 36, where each polyhedral side is represented by a two-dimensional array of boxes (chambers).

4.6.2.1 Detector structure

The detector builder relies on several parameters:

- General parameters:
  - `numSides`: The number of polyhedron sides.
  - `overlapY`, and `overlapZ`: The dimensions of the overlap between adjacent chambers in the two directions.
- Barrel parameters:
  - `numDetectorLayers`: The number of barrel layers.
  - `rmin`: The inner radius of the barrel detector.
  - `length`: The total length of the barrel detector along the z-axis.
  - `numYokes`, `yoke_Thickness`, and `yoke_Material`: Parameters defining the barrel iron yokes, i.e., the number of yoke layers, their thickness, and material, respectively.
- Endcap parameters:
  These are similar to the barrel parameters, with the following distinctions:
  - `rmin`: The inner radius of the endcap disks.
  - `rmax`: The outer radius of the endcap disks.
  - `z_offset`: The z-axis offset of the starting point of the endcap disks for both the positive and negative endcap layers.

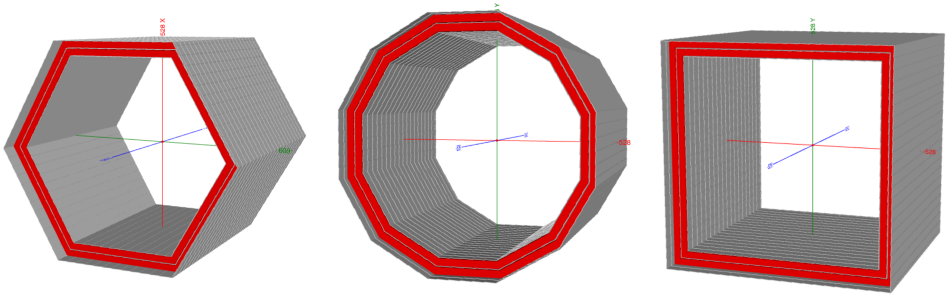


Fig. 37: Examples of different barrel polyhedron shapes that can be constructed by the builder. From left to right the `numSides` has takes the following values respectively: 6, 12, 4. The gray layers represent the sensitive detector layers, while the red layers represent the yokes.

The builder is very generic and its parameters are very flexible, allowing users to easily adapt it to their specific detector design. There are almost no place for hard-coded parameters. The building of the detector starts from defining the shape, and the user can determine that by changing `numSides` parameter, the shape can be cube, hexagon prism, octagon prism, etc, as shown in Fig. 37. Once the number of sides is specified, the builder subdivides both the barrel and the endcap into `numSides` parts, referred to as "sides," each represented by a two-dimensional array of chambers. The dimensions of each side are automatically computed based on `rmin` and `length` for the barrel, and `rmin` and `rmax` for the endcap. For example, in the barrel, each side is a rectangle with a length equal to the barrel's `length`, and a width computed as:

$$\text{side width} = 2 \cdot \texttt{rmin} \cdot \tan\left(\frac{180^\circ}{\texttt{numSides}}\right)$$

In the other side, the endcap side is trapezoid shaped, and its dimensions are determined automatically by using the endcap's `rmin` and `rmax`.

The sides itself are also overlapping, without touching each other, to make sure that the design has no cracks.

Every side is divided again into smaller overlapping rectangles with the same width as the chambers, and each rectangle is a one-dimensional array of overlapping detector chambers, resulting in the side structure shown in Fig. 36.

If there is no integer division of the side length into rectangles or of the rectangles into chambers, the builder will build chambers and rectangles with unusual dimensions to fit the remaining spaces in each side, an example is shown in the image on the right in Fig. 36.

The builder has the ability to make multiple layers, and the user can determine the number of the layers by changing the parameter `numDetectorLayers` for the barrel and and endcap detectors. The inner radius and the length of each layer is automated by the code taking into account they follow each others, with the final barrel layer enclosing the entire detector.

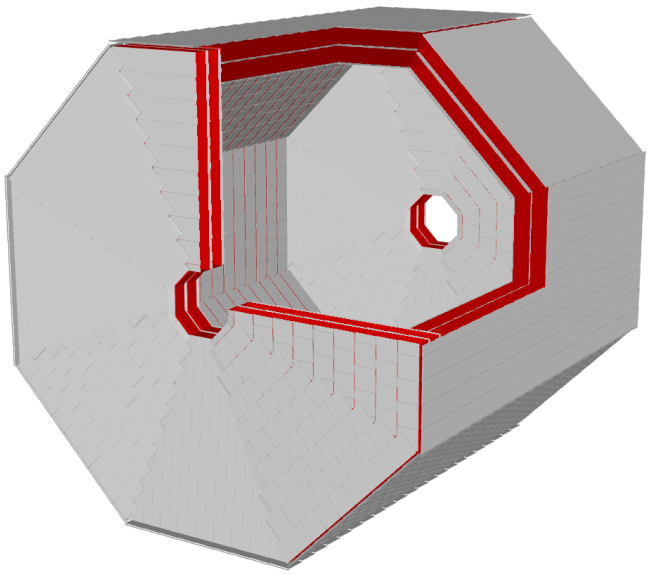


Fig. 38: IDEA muon system as an example of layered polyhedron with overlaps.

In addition to the sensitive layers, the builder can generate iron yokes (radiators) to slow down high-energy particles, such as muons. This approach is common in muon systems of multi-purpose experiments like CMS. If the users doesn't want to build yokes in their detector, they can easily put `numYokes` = 0. Otherwise, they can determine the number of the yokes they want to add through the same parameter. The thickness and the material of the yokes can be defined by the parameters `yoke_thickness` and `yoke_material` respectively. Building of yokes is much easier than the detectors, they are also divided into number of sides in each layer = `numSides` of the polyhedron shape. The barrel yokes are described on the shape of trapezoid along the z-axis, taking the full length of the detector. The angle of the trapezoid is the same angle of the detector shape (octagon, hexagon, ...). That will assure they will perfectly meet at the corner without intersections. The same for endcap yokes, but trapezoids are perpendicular to the z-axis. The builder is programmed to make every yoke layer follow a detector layer.

Overall, the builder can be used to describe both the barrel and the endcap, or one of them by setting the other's `numDetectorLayers` to 0. An example is shown in Fig. 38, illustrating the IDEA muon system, which is built using this tool. The IDEA muon system features three detector layers and two yoke layers in both the barrel and endcap, forming an octagonal prism.

4.6.2.2 Outlook

While the implementation of the builder is complete, there are some plans to make it more flexible and adoptable for broader applications. Here are some of the foreseen developments:

- Instead of building chambers with non-standard dimensions to fill side gaps, the approach will shift to using standard-size chambers throughout the detector. The overlap will be increased in the final chamber of each side to eliminate gaps using fully-sized chambers with larger overlaps.
- Splitting the builder into two separate tools — one for the barrel and one for the endcap — allowing independent collections of hits for each.

4.7 Full detector models

The availability of these sub-detector geometries along with the plug-and-play philosophy of DD4hep facilitates the implementation of various full detector concepts. The currently available configurations (CLD, CLD with noble liquid calorimeter, CLD with the ARC detector, IDEA with and without crystal calorimeter and ALLEGRO) are described in the main PED Feasibility Study Report document. Instructions on how to run the state of the art SIM-DIGI-RECO sequence for these detector models are maintained in GitHub repositories [87, 88]

5 Geant4 interface

This section describes the way Geant4 is currently interfaced in FCC full simulation (DDSim) together with the custom – i.e. not provided centrally by DD4hep – 'sensitive detector actions' that have been developed to meet the needs of FCC sub-detectors.

5.1 DDSim

DDSim is a simulation tool within the DD4hep suite [1, 89], primarily designed for detector physics simulations in high-energy physics experiments. Built on top of the Geant4 toolkit [11, 30], DDSim provides a user-friendly interface while integrating with the DD4hep detector description framework. It comes with default settings that are well-suited for most HEP applications. The only truly mandatory inputs are the so-called compact XML file, which defines the detector geometry, and the `runType`. Depending on the `runType` (e.g. `batch` mode), additional inputs, such as the number of events, may be required.

5.1.1 Configuration

The simulation in DDSim is configured using either a steering file, command-line arguments or both. Before starting the simulation, two key elements must be defined: the detector geometry and a physics list. The detector geometry is described by a DD4hep compact file. The baseline for hadronic and electromagnetic physics is typically one of the built-in Geant4 physics lists [30], being `FTFP_BERT` the default one. Additional modules, such as optical physics, can also be included as needed. DDSim automatically adds a step-limit physics module by default, ensuring that any user-defined limits specified in the geometry take effect. You can also configure Geant4 directly through UI commands, which can be provided via the steering file or a dedicated Geant4 macro file.

Event simulation begins with the creation of primary particles within the geometry. This can be done using particle guns, either the native DDSim gun or Geant4 guns configured via a macro file. Alternatively, an input file can be used to generate primary particles. DDSim supports various input formats, including .stdhep, .slcio, .HEPEvt, .hepevt, .pairs, .hepmc, and hepmc3. For accurate physical event simulations, vertex smearing, offset, and crossing angle boost can also be specified. The selection of primary particles and their pre-assigned decays is also handled by DDG4, to account for particles with a significant lifetime.

For each event, DDSim creates a Monte Carlo particle tree that includes all primary particles and any secondary particles produced within a designated area called the *tracker region* (unless it is called with the option `part.keepAllParticles` in which case all secondary particles are kept). This tree tracks both static details (like charge and mass) and dynamic properties (such as momentum and position) of particles at the start and end of their paths. Additionally, any recorded hits can be linked to the particles in this tree.

The steering file offers flexibility by allowing non-default sensitive detector types, actions, and filters to be associated with the detector. It also allows configuration of

the Geant4 tracker integrator and random engine behaviour. Finally, the output data format can be set to EDM4hep, DD4hep, or LCIO.

5.2 Custom sensitive detector actions

Each sub-detector may link certain volumes to a sensitive detector object, which determines the type of hit data that will be included in the output collection. A sensitive detector action, which can be specified in the steering file, is executed at every step within these volumes. To optimise performance, filters based on incoming particle properties, such as type or energy, can be applied to avoid unnecessary calls to the sensitive detector action. DD4hep provides predefined data types, actions, and filters for both types of sensitive detectors. However, custom functionality can be added using the plug-in system. An alternative option for scoring specific particle properties – such as energy, flux, current – are concrete primitive scorers [30], which can be enabled via the Geant4 UI or macro files. Primitive scorers in Geant4 are useful because they can overlap with the material geometry, allowing for the measurement of basic quantities without the need to define a dedicated detector.

5.2.1 Segmented crystal EM precision calorimeter (SCEPCal)

The SCEPCal is the only sub-detector that records both scintillation and Cerenkov photon signals in the same volume. A custom sensitive action class is used to save two separate readout collections, one for scintillation photons and one for Cerenkov photons. The standard edm4hep::SimCalorimeterHit class is used for each collection, with photon counts taking the place of energy deposit.

Optical photons may be terminated after the first count is recorded to save computation time. Alternatively, the scintillation yield of the crystal material can be adjusted without terminating the photons, as the scintillation yield is proportional to the energy deposit.

5.2.2 Dual readout capillary tubes calorimeter

A custom sensitive detector action for the capillary tubes dual readout calorimeter has been implemented as a k4geo plug-in to allow fast full-simulation of particles showering while preserving a good Monte-Carlo-to-data agreement. This solution is inspired by the sensitive detectors currently used in production for optical calorimeters at the LHC Experiments.
The signal produced in scintillating fibres is computed starting from the energy deposit of each step of electrically charged particles in such fibres. This is the seed for the signal calculation. Each energy deposition is transformed into the number of optical photons via a light yield conversion (scintillating-photons/MeV) as measured in beam testing. Additionally, a Poisson smearing on the amount of optical photons is applied at each step, to reproduce the scintillation fluctuations in light emission. At each step, the distance between the step position inside the optical fibre and the fibre tip readout by the SiPM is calculated, and the amount of light is suppressed by an exponential decay of light based on the optical fibre attenuation length. Finally, a Binomial survival probability is applied to estimate the number of photo-electrons detected by the

SiPM according to its photo-detection-efficiency.
To calculate the Cherenkov signal in clear plastic fibres, the procedure must consider additional information about the directionality of the Cherenkov photons. Each charged particle step in a clear fibre generates Cherenkov optical photons, as calculated by the Geant4 Cherenkov process. Such photons are emitted with the typical Cherenkov directionality. Depending on the emission angle and the fibre core and cladding refractive indices, they are trapped inside the fibre or escape the fibre core. To correctly reproduce it in the simulation, Cherenkov optical photons are included as Geant4 tracks, and once they reach the core-cladding boundary Geant4 calculates the corresponding boundary process, i.e. whether a photon is reflected inside or escapes the fibre. Only the photons trapped inside are considered when calculating the Cherenkov signal. An additional check calculates whether a trapped photon is travelling towards the fibre free-tip or the sipm-readout tip. Only those travelling towards the SiPM are considered as contributing to the fibre Cherenkov signal. Once the number of Cherenkov photons trapped inside the fibre and travelling towards the readout is calculated, an exponential attenuation of light is applied, similarly to the scintillation case, based on the fibre attenuation length and the distance between the step and the sipm-readout fibre tip. Finally, to calculate the number of photons detected by the SiPM, i.e. the number of photo-electrons, a Binomial survival probability is applied according to the photo-detection-efficiency.
These methods allow calculating at each step the signal produced inside an optical fibre, and the total signal recorded in each fibre per event is the sum of each step's contribution.
This custom sensitive detection action achieved an excellent event rate. For 10 GeV electron events, it is between 0.7 and 1.0 seconds per event, with a small dependence over the showering point inside the calorimeter geometry, and the event rate scales linearly with the primary particle energy. At the same time, it allowed precise and detailed comparison with test-beam data leading to a good Monte-Carlo-to-data agreement [90].

6 Overlay, digitisation and reconstruction

After having simulated the passage of particles through the detector and collected the energy deposits from Geant4, the response of the detector electronics is emulated and high-level quantities are built to serve various physics purposes including detector performance estimation, detector optimisation and physics analyses. This section starts by describing how the contributions from beam induced background are overlaid with physics events and then covers the various digitisation and reconstruction algorithms centrally available in the FCC software. Unless stated otherwise, all those algorithms are relying on the Gaudi framework and receive/output EDM4hep objects.

6.1 Background overlay

`OverlayTiming` is an algorithm originally developed by the Linear Collider community that has been ported to Key4hep. The purpose of this algorithm is to combine background and physics events into single events.

The overlay algorithm processes both physics and background events read from files, treating each collection differently:

- For the `edm4hep::MCParticle` collection, all particles from the physics event are included without filtering or modifications. Particles from background events are overlaid with a new time, offset from their original time.
- For the `edm4hep::SimTrackerHit` collection, both physics and background hits are overlaid only if they fall within a user-defined time window.
- For the `edm4hep::SimCalorimeterHit` collection, physics event hits are included only if they have any `edm4hep::CaloHitContribution` within a specific time window. In such cases, only the contributions within this window are considered. For background hits, if a hit already exists for the `cellID` of the background hit, only the contributions from the background hit within the time window are added. If no hit exists with the background hit's `cellID`, a new hit is created (with attributes cloned from the background hit) with the contributions that fall within the time window.

The Overlay algorithm offers several configurable parameters. Users must select background files, and can create different groups of background files with distinct parameters (e.g., when overlaying various background sources). The number of background events to overlay can be determined using either a Poisson distribution or a fixed number. Users can also specify a time offset for background `edm4hep::MCParticle`s and set time windows for accepting `edm4hep::SimTrackerHit`s and `edm4hep::CaloHitContribution`s.

Furthermore, special attention has been given to ensure full utilisation of the overlaid collections. In PODIO, when an object is cloned, the relations of the new object typically point to the old object. To address this, functionality was developed to make the overlaid collections consistent, ensuring they point to objects within the overlaid collections rather than the original ones. This allows for a complete transition to the new collections after overlaying.

6.2 Planar tracker digitisation

The FCC software provides several algorithms emulating the digitisation of energy deposits in trackers (`edm4hep::SimTrackerHit`) with planar sensitive surfaces. So far, they mostly rely on a simple parametrisation of the spatial resolution through Gaussian smearings but algorithms with a more detailed treatment (e.g. including the charge sharing effect) are under development, as described in Section 6.14. Though they are typically applied to silicon trackers, some of those algorithms can also be applied to other types of sub-detectors (e.g. muon systems) provided that they provide `edm4hep::SimTrackerHit` objects as output.

`DDPlanarDigi` is a digitiser previously used by the Linear Collider community. This digitiser has been ported to Key4hep as a Gaudi Functional algorithm. It takes a collection of `edm4hep::SimTrackerHit` as input and outputs a collection of `edm4hep::TrackerHitPlane` along with a collection of links between the input and output hits.

The algorithm processes each input hit, attempting to smear its position within the plane of the measurement surface, obtained through the `cellID` of the hit. If a hit falls outside the sensitive volume, the algorithm will make a configurable number of attempts to smear its position again. Once a valid position inside the sensitive volume is achieved, an output hit is generated. This output hit retains most of the original input hit's data, with potential modifications to its time (if configured) and position. If the number of attempts exceeds the maximum allowed, the hit is discarded, and no corresponding output hit is produced. Configurable parameters include the width of the Gaussian distribution used for position smearing, whether to smear the time, a time window for hit selection, the maximum number of positioning attempts, and the minimum energy threshold for hit acceptance.

The algorithm generates multiple histograms containing data about the hits, such as energy, as well as information about the algorithm's performance, including the extent of position smearing for accepted hits and the fraction of hits accepted. Validation has been performed to ensure that the results match those of the original algorithm. Additionally, the ported algorithm has been successfully used for full simulation with CLD, facilitated by the Marlin wrapper.

A second, separate algorithm to digitise semiconductor sensor hits was developed in Gaudi, with similar features to `DDPlanarDigi`. It is hosted in the k4RecTracker repository[3], which also contains drift chamber digitisation as discussed in the following sections.

6.3 Drift chamber digitisation

A first simple algorithm was developed to emulate the digitisation of hits produced by the simulation of the full stereo drift chamber in a parametrised way. This algorithm handles two main tasks: smearing the hit positions and estimating the cluster count needed for particle identification (PID). The user defined smearing occurs in the wire's coordinate system, both along the wire and perpendicular to it. The data extension attached to the DD4hep geometry of the detector is used to calculate the wire positions

[3] https://github.com/key4hep/k4RecTracker/tree/master

without needing the actual wire volumes. The process for determining the number and size (number of electrons) of ionisation clusters follows the method described in this reference [91]. A more detailed simulation of the digitisation process, including full waveform analysis to better estimate the impact of background events, will be added in the future.

Both tasks require the generation of random numbers, so several precautions were taken to ensure reproducibility and thread safety:

- Before processing each event, a unique seed is generated by the `IUniqueIDGenSvc` service, based on the run number, event number, and the algorithm's name.
- Random engines are defined as class members and marked as `thread_local`.
- Random distributions are defined as class members and marked as `mutable`, allowing their use within a `const` function.

Additionally, a data extension of EDM4hep was developed to store the digitised hits. The existing class for digitised hits in silicon trackers could not be used due to the absence of fields for storing cluster count information and due to the fact that, before tracking, no 3D position can be defined for drift chamber hit (only the distance to the wire is known). As work progresses, it may be necessary to extend the class, for example, by adding a new field to store the covariance matrix.

6.4 Generic calorimeter digitisation from ILCSoft

The `DDCaloDigi` processor from the ILCSoft DDMarlinPandora package [92] provides a generic calorimeter digitiser able to handle a per layer sampling fraction, an analogue or digital mode (in the latter, the energy is estimated based on the number of hit instead of the total energy) and a zero-suppression emulation. This digitiser is used for the CLD detector through the Marlin wrappers and lcio $\leftrightarrow$ EDM4hep converters but its detailed description is outside of the scope of this note.

6.5 Key4hep native generic calorimeter digitisation

The above-described ILCSoft digitiser was not fitting all the needs of e.g. the Noble Liquid calorimeter, where noise and cross-talk effects have to be studied. A new version of generic calorimeter digitiser was therefore implemented as 'native' Gaudi algorithms based on the EDM4hep data model in the `k4RecCalorimeter` package [93]. This new version handles a per layer sampling fraction, noise addition, cross-talk emulation and zero-suppression but only treats the analogue mode.

This digitisation uses the `CreatePositionedCaloCells` class. A `SimCalorimeterHitCollection` (parameter hits) containing the hits created by Geant4 in the active volume of the calorimeter is passed as input to the class. Hits with the same `cellID` are combined into a single cell, whose energy equals the sum of the individual hit contributions. Optionally, noise can be added to the cell energy, by setting the `addCellNoise` parameter to True (default: False) and providing a `noiseTool` (see Section 6.5.1 for more details). Cross-talk effects can also be included in the digitisation, by setting the `addCrosstalk` parameter to True (default: False) and providing a `crosstalkTool`, as described in Section 6.5.2. The cell energy is

calibrated to the electromagnetic energy scale by setting the `doCellCalibration` parameter to True and passing to the algorithm via the `calibTool` an instance of a `CalibrateInLayersTool`. The latter provides a list of sampling fractions, one per longitudinal layer of the calorimeter, determined from simulations of 10 GeV single electrons in which also the passive elements of the calorimeter are turned into active elements: the total energy deposited by the electrons in each layer is recorded, and the ratio between that deposited in the LAr and the total is used to calculate the sampling fraction. The position of the peak of a Gaussian fit to the scale factor in each layer is used as the nominal sampling factor. The sampling fractions have been confirmed to be independent of the particle energy and direction. Simulated 10 GeV photons yield sampling fractions that agree with the nominal ones from electrons within 2–3%, where the statistical uncertainty from the sample size (10000 events) is at the 1% level. This small difference is absorbed by the dedicated particle energy calibration performed at a later stage. The cells produced by the digitiser algorithm are then saved into the output `CalorimeterHitCollection` whose name is set with the `cells` property of `CreatePositionedCaloCells`.

The cell object stores the calibrated energy and `cellID` of a given readout cell. To ease downstream reconstruction algorithms, the cells are also assigned three-dimensional coordinates based on the position of their central point. This task is performed by a positioning tool, that takes care, based on a given segmentation, of translating the information encapsulated within the `cellID` into a space point. The tool, which implements the `ICellPositionsTool`, must be configured and then passed to `CreatePositionedCaloCells` via its `positionsTool` property.

In the case of the noble liquid electromagnetic barrel calorimeter, the tool is an instance of the class `CellPositionsECalBarrelModuleThetaSegTool`. The tool is informed with properties of the segmentation used for the readout of the detector (`cellID` encoding format, number of layers, number of adjacent modules and θ cells grouped together in each layer, and so on) and uses them to decode from the `cellID` information such as the identifier of the Geant4 volume to which they belong to – from which the radial position of the cell is inferred – and layer, module and θ index of the cells, from which the other two spatial coordinates are calculated.

6.5.1 Noise emulation

Noise effects can be included in the digitisation by setting the option `addNoise` of `CreatePositionedCaloCells` and passing to the digitisation algorithm a noise tool. The latter implements the `INoiseCaloCellsTool` interface and returns the expected noise offset and rms for a cell based on its `cellID`. The noise constants should take into account all expected sources of noise (e.g. from electronics and pile-up). When noise is emulated, the user also needs to provide to the digitiser a `geometryTool` conforming to the `ICalorimeterTool` interface, that provides the `prepareEmptyCells` method. This method is used within the digitiser to create a map containing the full list of cells in the calorimeter, with zero initial energy. During the processing of each event, after the calculation of the energy deposit and the optional application of cross-talk effects, the digitiser code generates for each cell in the map prepared by the `geometryTool`,

a random noise value according to the expected noise offset and standard deviation (assuming a Gaussian distribution), and adds it to the signal (if any) of the cell. Furthermore, if the `filterCellNoise` option is set in `CreatePositionedCaloCells`, cells with energy (or absolute value of the energy, to avoid biasing the energy of the clusters) below a multiplier of the expected noise are removed from the output cell collection, before they are passed to the clustering algorithms.

A concrete Gaudi tool implementing the `INoiseCaloCellsTool` interface is `NoiseCaloCellsVsThetaFromFileTool`, used in the digitisation of the ALLEGRO noble liquid electromagnetic barrel calorimeter. The tool reads, from an external ROOT file, histograms of pre-calculated expected electronic and pile-up noise constants as a function of polar angle for each longitudinal layer, as shown in Fig. 39, and returns, for a cell of given `cellID`, the total expected noise offset and standard deviation based on its layer index and polar angle.

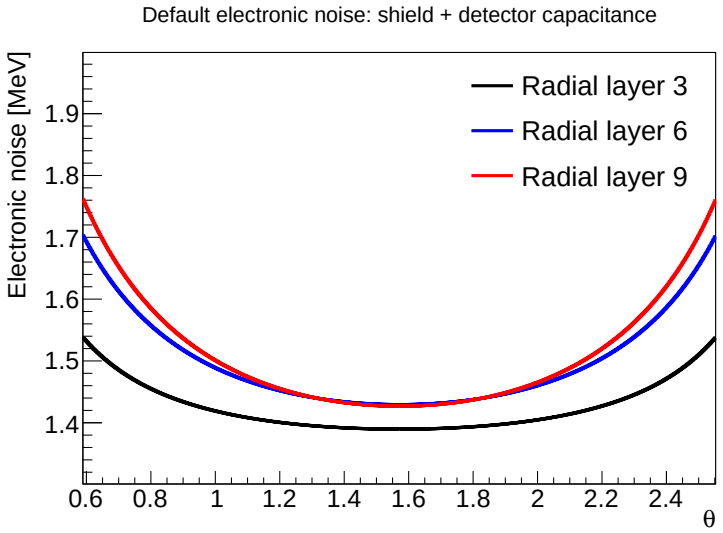


Fig. 39: The estimated level of electronic noise as a function of θ (in radians) for 3 different radial layers. The outer layer is labelled with a larger index. The size of cells increases with the layer index and the θ-position.

6.5.2 Cross-talk emulation

The effect of cross-talk is one of the main challenge to implement the high granularity design of the Noble Liquid ECAL readout electrodes. The cross-talk in this calorimeter refers to the phenomenon where the current generated by the energy deposit of an EM shower in a given readout cell also induces a current signal in neighbouring cells due to various electromagnetic couplings (capacitive, inductive and resistive). For a given granularity of the detector, the design of the readout electrode is a delicate balance between the cross-talk and electronics noise. Specifically, while a reduced shielding

lowers the capacitance to the ground and therefore lowers electronics noise, stronger cross-talk effect may appear because of less effective shielding.

The presence of cross-talk degrades the detector performance in general, but its impact can in particular affect, depending on the magnitude of the cross-talk, the reconstruction algorithms and the performance in certain specific areas. For example, in an EM calorimeter, cross-talk can play a critical role in both the spacial resolution and particle identification (PID), while having on the other hand minor impact on the energy resolution. Even though the effect of cross-talk can be measured and partially corrected for during data reconstruction, a dedicated emulation study is a crucial task in the R&D phase of the detector design, to help optimising the design of readout electrodes in terms of granularity, noise and cross-talk, and to provide a better understanding of the cross-talk tolerance for the performance.

The cross-talk emulation described in this section is developed for the barrel region of the ALLEGRO ECAL. The principle of the emulation, however, can be readily generalised to other detector geometries. Given the multi-layer PCB design of the ALLEGRO ECAL, the study of cross-talk emulation accounts for four types of cross-talks as described here-under. The classification of cross-talk types are explained in terms of neighbouring cells in Fig. 40, which presents part of a ϕ module of the readout panel on ALLEGRO ECAL. Cross-talk is only considered within the same readout panel. The segmentation of the considered readout panel consists of radial layers and θ towers. The four types of cross-talk are:

- Direct radial neighbours. They are the cells directly on the inner or outer radial border of the cell under study inside the same θ tower. The cross-talk to the inner and outer radius cells, coming from pad-pad and trace-pad couplings, is expected to be of similar magnitude due to symmetries.
- Direct θ neighbours. Similar to the aforementioned direct radial neighbours, direct θ neighbours concern the calorimeter cells sharing the same index of the radial layer with the cell under study, but with adjacent θ index.
- Diagonal neighbours. Despite the small magnitude of cross-talk existing in this scenario compared with the first two types, cells with both ± 1 difference in the radial index and ± 1 difference in θ are still considered to be cross-talk neighbours for a complete setup of the emulation.
- All other cells contained in the same θ tower, with more than ± 1 difference in the radial index. Considering that the readout line of a cell runs beneath other cells within the same θ tower, a capacitive cross-talk can occur to all cells in this θ tower in case of limited shielding within the PCB. In the emulation, all signal transmission lines are assumed to be extracted from the outer radius side of the readout electrode, which justifies the choice of using a universal value to represent the magnitude of the capacitive cross-talk. This capacitive cross-talk between cells with ± 1 difference in the radial index of the same θ tower is already accounted for in the "direct radial neighbours" cross-talk to avoid double-counting.

One value of cross-talk coefficient is assigned to each type of cross-talk neighbour in the emulation. It has to be pointed out that the same set of cross-talk coefficients applies to the strip layer as well, regardless of a 4 granularity times finer in the θ

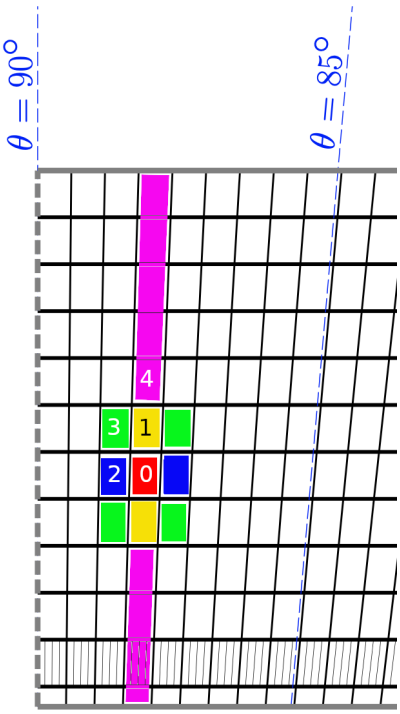


Fig. 40: Cross-talk neighbours of ALLEGRO ECAL: (0) Reference cell. (1) Direct radial neighbours. (2) Direct θ neighbours. (3) Diagonal neighbours. (4) θ-tower neighbours.

Table 10: Cross-talk coefficients tested in the emulation.

	Direct radial	Direct θ	Diagonal	θ tower
No cross-talk	0	0	0	0
Cross-talk with 50 ns shaping	0.7%	0.2%	0.04%	0.1%

direction. Also, the definition of θ tower neighbours contains all four strips in the same θ tower.

In order to carry out a realistic study of the cross-talk effect, scenarios with and without cross-talk are tested in the emulation as a comparison. Values of cross-talk coefficients are detailed in Table 10. All cross-talk coefficients in the setup with cross-talk are taken from a measurement in the laboratory on an electrode prototype, with a pulse shaping of 50 ns.

For the technical implementation, the emulation is separated into two steps. In the first step, a map of cross-talk neighbours is generated for a given geometry of the ALLEGRO ECAL. The map contains indices of all calorimeter cells, their cross-talk neighbours for each type, as well as the corresponding cross-talk coefficients

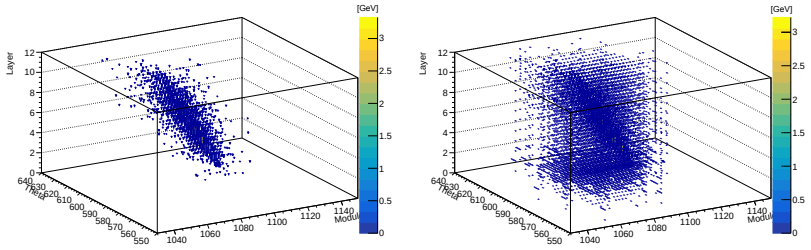
between a given cell and its cross-talk neighbours. In the second step, this cross-talk map is fed to the calculation of cell energies deposit in the full simulation via the `ReadCaloCrosstalkMap` cross-talk tool passed to `CreatePositionedCaloCells`. The tool implements the `ICaloReadCrosstalkMap` interface and returns, for a cell with given `cellID`, the list of cross-talk coefficients. In the digitiser, the energy deposit due to the EM shower in each cell is recorded as a baseline and re-distributed to all its cross-talk neighbours according to the cross-talk map. Therefore, the final energy of a single cell is equal to the energy deposit in the EM shower, subtracted by the energy it gives to cross-talk neighbours, then added by the amount of re-distributed energy deposit it receives from all its cross-talk neighbours. Since the reconstruction of calorimeter clusters only takes final cell energies as input, the modification of cell energies due to the cross-talk is automatically propagated to the downstream cluster reconstruction algorithms. The loops of reciprocal computation of energy re-distribution in the cross-talk emulation is very CPU consuming. Consequently, the emulation of cross-talk effect is only enabled in the full simulation when performing a dedicated study of cross-talk.

The energy resolution and ϕ resolution of ALLEGRO ECAL are studied when considering the effect of cross-talk. Twenty thousand events corresponding to 50 GeV electrons are injected to the ALLEGRO detector from the interaction point located in the geometric centre of the detector. The response of the detector is calculated from the CaloTopoCluster, before and after taking into account the cross-talk effect in the full simulation. Signals of calorimeter cells used as input to the reconstruction of CaloTopoCluster are shown in Fig. 41 without and with cross-talk. Without adding any filter, a clear spread of signal is visible in Fig. 41 (b) compared with Fig. 41 (a) due to the effect of cross-talk. However, with a filter of about 1σ level of average electronic noise (0.5 MeV on the first layer and 1.5 MeV on the other layers), it can be appreciated from Fig. 41(c) and (d) that only marginal spread of signals is observed in the core part of the EM shower. The energy and ϕ response are shown in Figures 42 and 43, respectively. No noise filter is applied to the calculation of energy and ϕ responses. No significant degradation is seen for either of the two responses. Further studies of the cross-talk impact on shower shape variables are planned to investigate the impact on PID.

6.6 Silicon photomultiplier digitisation

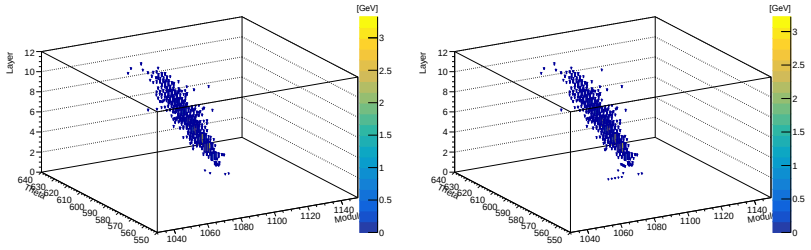
Silicon photomultipliers (SiPMs) are solid-state light sensors with high photon detection efficiency (PDE), excellent timing resolution and magnetic field immunity. They benefit from the rapid evolution of silicon technology and, due to their compactness, ruggedness and low cost, can be considered an attractive alternative to traditional Photomultiplier tubes (PMT). For this reason, many detector concepts and sub-detector systems consider SiPMs as a major candidate for detecting photons [94].

Typical area of commercially available sensors goes from 1×1 mm^2 to 6×6 mm^2 with micro-cells pitch ranging from 10 µm to 100 µm. A SiPM can be seen as an high-density array of independent micrometer-sized pixels each operated as a Single-Photon Avalanche Diode (SPAD). When a single electron is released in the micro-cell by an incoming photon, it generates an avalanche with a multiplicative factor of the order of 10^6, producing a detectable signal. All micro-cells are placed on the same



(a) No cross-talk, no filter

(b) Cross-talk with 50 ns pulse shaping, no filter



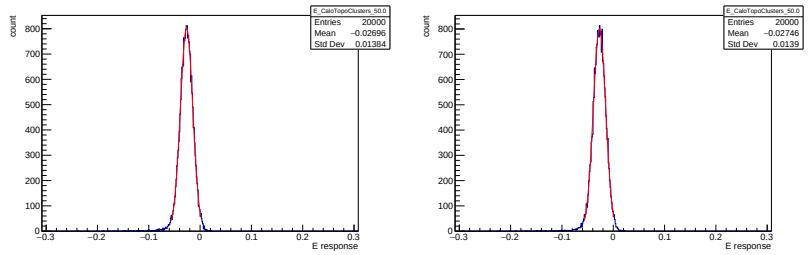
(c) No cross-talk, after filter

(d) Cross-talk with 50 ns pulse shaping, after filter

Fig. 41: Signals generated in ALLEGRO ECAL cells by a 50 GeV electron before and after adding cross-talk. The signal for each cell is computed as the sum of energy deposit by the EM shower and total effect of cross-talk, then divided by the sampling fraction of the corresponding layer. While no filter is added to cells in (a) and (b), only cells passing the 1σ electronic noise filter are shown in (c) and (d).

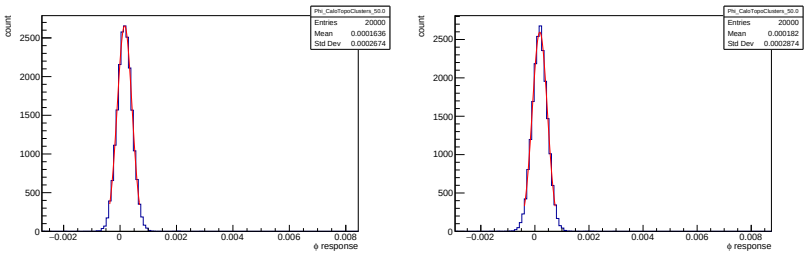
substrate and connected in parallel, therefore the signal generated is proportional to the number of fired cells. The response of the SiPMs is affected by some side effects such as spurious signals (dark count rate), after-pulsing and crosstalk. All effects are well studied and documented, they usually depend on the characteristics of the sensor and the technology in use and can be easily retrieved from product data-sheets or measured in laboratory. Another side effect is the non-linear response to incoming light. It is due to the limited number of pixels available per mm^2 and may affect applications where high intensity light is expected. In addition, there are applications that want to use the entire signal generated by the SiPM. In this case, it may not be sufficient to add signals to the white noise to describe the detector response.

Therefore, a robust simulation that includes all these effects is important to select the optimal sensor for a given application, helping to assess the impact of all these effects to the final performance. Indeed, a SiPM emulation library, SimSiPM, has been developed to describe in detail the SiPM's waveform [95]. It can also be used to perform optimisation studies considering different SiPM models, which allows the choice of the most suitable product available on the market. It is worth mentioning that the accuracy of the simulation strongly depends on the inputs. The simulation is



(a) No cross-talk. (b) Cross-talk with 50 ns pulse shaping.

Fig. 42: Energy response of ALLEGRO ECAL for topo-clusters, before and after introducing cross-talk. No noise filter is applied. The energy response is calculated as $(E_{reco} - E_{true})/E_{true}$.



(a) No cross-talk. (b) Cross-talk with 50 ns pulse shaping.

Fig. 43: ϕ response of ALLEGRO ECAL for topo-clusters, before and after introducing cross-talk. No noise filter is applied. The ϕ response is calculated as $\phi_{reco} - \phi_{true}$.

designed to use parameters that can be retrieved in the company data sheets or that can be easily measured in the laboratory.

The properties of a SiPM are described by the class `sipm::SiPMProperties`. Once the SiPM properties are configured, the class `sipm::SiPMSensor` can be constructed with the `sipm::SiPMProperties` instance as an input to the constructor. Incoming photons are characterised by their arriving time, which can be sorted into `std::vector<double>` and put into `sipm::SiPMSensor::addPhotons(const std::vector<double>&)`. If the user wants to utilise the PDE, an additional `std::vector<double>` filled with the photon's wavelength can be provided to `sipm::SiPMSensor::addPhotons`. Lastly, the output signal waveform can be generated by `sipm::SiPMSensor::runEvent()` and retrieved using the method `sipm::SiPMSensor::signal()`.

The output signal waveform is described by the class `sipm::SiPMAnalogSignal`, and it equips several functionalities to extract features similar to what conventional analogue-to-digital converter provides, including the integral of the signal, height of

```
1
2  // Create sensor and set parameters
3  SiPMProperties myProperties;
4  SiPMSensor mySensor(myProperties);
5  // ...
6
7  // Store results in here
8  std::vector<double> integral(NEVENTS);
9  // peak
10 // ...
11
12 for(int i=0;i<NEVENTS;++i){
13   // Generate photons times accordingly
14   // to your experimental setup
15   mySensor.resetState();
16   mySensor.addPhotons(times);
17   mySensor.runEvent();
18
19   SiPMAnalogSignal mySignal = mySensor.signal();
20
21   integral[i] = signal.integral(10,250,0.5);
22   // peak
23   // ...
24 }
```

Listing 1: An example event loop of SiPM signal generation.

the peak voltage, and the time of arrival or peak. The bare waveform of the signal can be returned as a `std::vector<float>` by `sipm::SiPMAnalogSignal::waveform()`. An example of the complete event loop can be seen in the below.

The SimSiPM library is available in the Key4hep software stack via Spack. A Gaudi algorithm `DigiSiPM` is developed under the `HEP-FCC/dual-readout` repository to emulate SiPM for the Dual-Readout calorimeter. The custom sensitive detector action of the Dual-Readout calorimeter creates the number of detected optical photons per sensitive element with `edm4hep::RawCalorimeterHit` and its distribution as a function of time using `edm4hep::RawTimeSeries`. A `std::vector<double>` length equals the number of optical photons is created with the recorded time as its elements. The SiPM emulation is initiated through `sipm::SiPMSensor::addPhotons` using the vector as an input. The sensor characteristics and parameters for charge integration are configurable with `Gaudi::Property`. Lastly, the output integrated charge and waveform are stored in `edm4hep::CalorimeterHit` and `edm4hep::TimeSeries`, respectively.

6.7 Tracking

The FCC software provides two algorithms for track reconstruction: a conformal tracking-based algorithm which can be applied to all-silicon trackers and a machine learning based-track finding which is more general and can be applied to different sub-detectors. Additional solutions suitable for drift chamber track fitting, such as the Deterministic Annealing Filter [96], are being investigated.

6.7.1 Conformal tracking

The conformal tracking and fitting methods used e.g. for the CLD detector have already been described elsewhere [97]. The original ILCSoft implementation is being ported to Key4hep.

6.7.2 Machine learning-based tracking

`GGTF_tracking` is a Key4hep Gaudi Functional algorithm that performs track finding. It takes as input the collections from the digitised hit collections from the different sub-detectors in the tracker and outputs a collection of `edm4hep::TrackCollection`. At the moment of writing this note, the algorithm is implemented for the IDEA detector, with inputs: `edm4hep::TrackerHit3DCollection` and `edm4hep::DriftChamberDigiCollection`, and it will be extended to be used with inputs from the CLD detector. However, a Python version has already been tested for both detectors and the performance evaluation is available in a dedicated note [98].

The algorithm processes each input hit into a geometric algebra representation and uses a pre-trained graph neural network architecture and a clustering algorithm (dbscan) to obtain a label for each hit. The different labels result in the set of reconstructed tracks. The graph neural network is trained in Python and exported as an ONNX model. Configurable parameters include the dbscan algorithm parameters: epsilon, the maximum distance between two output hits to be considered in the same neighbourhood, and min samples, the number of samples considered a core point in the clustering. The method is detailed in a dedicated note [98].

The GGTF evaluation is implemented as a `k4FWCore::MultiTransformer`: `GGTF_efficiency`. The inputs to the evaluation algorithm are the output track collection `edm4hep::TrackCollection`, the links to the simulated hits `edm4hep::SimTrackerHitCollection` and the MC particle collection `edm4hep::MCParticleCollection`. The output is a table of features and the number of fakes. The table of features has the following attributes:

- Assigned track: index of the track to which the particle has been assigned (0 if not assigned)
- isReconstructable: boolean value which is 1 if the particle can be reconstructed and 0 otherwise.
- isAssigned: boolean value which is 1 if the particle is assigned and 0 otherwise
- Purity: purity of the track to which the particle has been assigned (-1 if not assigned)
- Efficiency: efficiency of the particle concerning the track to which the particle has been assigned (-1 if not assigned)

6.8 Calorimeter clustering

Several clustering algorithm are implemented in order to group together calorimeter cells with non-zero energy that are sufficiently close to each other.

6.8.1 Sliding-window, fixed-size clustering

Sliding window clustering is performed by the `CreateCaloClustersSlidingWindowFCCee` algorithm. The algorithm builds rectangular-shaped clusters of cells and stores them in the collection specified via the `clusters` property. If the property `attachCells` is set to True, a collection of clustered cells is also created, whose name is configured via the `clusterCells` property.

The input to the algorithm are "towers" of cell energies computed by an instance of the `CaloTowerToolFCCee` tool. Various $\theta \times \phi$ windows (in units of towers) are used by the algorithm and are configured via the following properties:

- `nThetaWindow`, `nPhiWindow`: number of towers along θ and ϕ used to find seed clusters;
- `nThetaFinal`, `nPhiFinal`: number of towers along θ and ϕ used to build the final clusters;
- `nThetaPosition`, `nPhiPosition`: number of towers along θ and ϕ used to calculate the barycentre of the seed clusters;
- `nThetaDuplicates`, `nPhiDuplicates`: size of a $\theta \times \phi$ grid used to check for and remove duplicate clusters.

The `CaloTowerTool` can build towers of cells originating from several subsystems, such as barrel or end-caps of either electromagnetic or hadronic calorimeters. For each subsystem a separate collection of positioned, digitised cells can be supplied, together with the name of the corresponding segmentation class used for the digitisation. The tool main parameters also include the sizes of the towers along θ and ϕ, `deltaThetaTower` and `deltaPhiTower`. For the baseline simulation of the barrel electromagnetic LAr calorimeter, the size of the tower is equivalent to that of the readout cells in the non-strip layers: $\Delta\theta \approx 0.56^{\circ}$, $\Delta\phi \approx 0.008$. The tool first determines the extent of the various subsystems in θ and ϕ to calculate the total number of towers in a grid that covers all the subsystems, and then fills each tower with the sum of the transverse energies ($E * \sin(\theta)$) of the cells belonging to them, i.e. whose centre is within the $\Delta\theta \times \Delta\phi$ region spanned by that tower.

The `CreateCaloClustersSlidingWindowFCCee` algorithm works in the following way.

1. Towers are created by invoking the `buildTowers` method of the `CaloTowerTool`.
2. For each window of size `nThetaWindow` times `nPhiWindow` in units of towers, the sum of transverse energies of the towers is calculated. If this sum is above a given threshold (property `energyThreshold` of the algorithm) and it is a local maximum (i.e. larger than the energies of the windows centred on points shifted by ± 1 along either the θ or ϕ directions), then the group of towers in the window is considered a "seed" or pre-cluster candidate and added to the pre-cluster list. The barycentre of each pre-cluster is then calculated as the energy weighted position of its cells, within a smaller window of size `nThetaPosition` times `nPhiPosition` around the cluster geometrical centre, if the energy in this core region is above a certain fraction (property `m_energyThresholdFraction`, default value: 25%) of the minimum

transverse energy threshold (i.e. 10 MeV). If the energy in the core region is not enough, then the position of the pre-cluster is calculated from the energy-weighted mean of all cells in the pre-cluster. The pre-cluster transverse energy is then recalculated after centring the sliding window on the tower containing the pre-cluster barycentre, in a window of size `nThetaFinal` times `nPhiFinal`. If the transverse energy is below threshold, the pre-cluster is dropped.

3. The pre-clusters are sorted in descending order according to their transverse energy.
4. Duplicate pre-clusters are removed: a loop over the pre-clusters is performed, and if two pre-clusters are found next to each other (within a window of size `nThetaDuplicates` times `nPhiDuplicates`, the pre-cluster with lower energy is removed.
5. The final clusters are built from the remaining pre-clusters. The cluster energy is calculated dividing the transverse energy by $\sin(\theta_{\mathrm{cl}}$, where the cluster polar angle θ_{cl} is determined from its barycentre. If the property `energySharingCorrection` is True (False by default), and a tower is common to more than one cluster, then its energy is assigned to the various clusters proportionally to the energies of the clusters in the towers that are not shared.

In the simulations of the LAr electromagnetic calorimeter for ALLEGRO, the settings used for clustering the cells in the barrel detector are:

- `nThetaWindow`=9, `nPhiWindow`=17;
- `nThetaFinal`=9, `nPhiFinal`=17;
- `nThetaPosition`=5, `nPhiPosition`=11;
- `nThetaDuplicates`=7, `nPhiDuplicates`=13;
- `energyThreshold`=0.040 (GeV);
- `energyThresholdFraction`=25%.

6.8.2 Topological, variable-size clustering

Topological clustering is performed by the `CaloTopoClusterFCCee` algorithm. The clusters are built by grouping together in three-dimensional space neighbouring cells with energy-to-noise ratio above a given threshold, starting from a seed cell.

The algorithm main properties are the following:

- `clusters`: name of the cluster collection to create in output
- `clusterCells`: name of the clustered cells collection to create in output
- `TopoClusterInput`: instance of a `CaloTopoClusterInputTool` tool;
- `neighboursTool`: instance of a `TopoCaloNeighbours` tool;
- `noiseTool`: instance of a `TopoCaloNoisyCells` tool;
- `seedSigma`, `neighbourSigma`, `lastNeighbourSigma`: thresholds in unit of expected σ_{noise} used to identify seed cells or neighbouring cells to be merged into the cluster

The `TopoClusterInput` tool collects the names of the positioned cell collections used for clustering, and the names of the corresponding segmentation (readout) classes.

The `neighboursTool` provides to the topo-clustering algorithm, for each cell identified by a given Cell ID, a vector containing the `cellID` of all cells that are considered as neighbours of that cell. The information is read from a `TTree` within a `ROOT` file created

with the `CreateFCCeeCaloNeighbours` tool. Within a given sub-detector, adjacent cells along the module (or ϕ) or θ directions are considered as neighbours; cells with only a corner in common are considered as neighbours if the `includeDiagonalCells` property is True (default: False). Cells of the same sub-detector in adjacent layers are considered as neighbours if they overlap at least partially in space. To create clusters of both ECAL and HCAL cells corresponding to hadronic showers, the property `connectBarrels` of the `CreateFCCeeCaloNeighbours` tool should be set to True. In that case, cells in the outermost layer of the ECAL are considered as neighbours of the innermost layer of the HCAL if the $\Delta\theta \times \Delta\phi$ region that they span partially overlap.

The `noiseTool` provides to the topo-clustering algorithm, for each cell identified by a given `cellID`, the offset noise and sigma. The information is read from a `TTree` stored in an external `ROOT` file.

The topological clustering algorithm works in the following way.

1. The cluster seeds are defined as all cells exceeding the signal/noise ratio that is given by `seedSigma`.
2. The vector of seeds is sorted according to their energies in descending order.
3. Proto-clusters are built by adding the neighbouring cells to the seed in case their signal/noise ratio is larger than `neighbourSigma`.
4. The procedure is repeated iteratively, adding to the proto-cluster the neighbours of the cells added in the previous iteration if their signal/noise ratio is larger than `neighbourSigma`, until no more cells exceeding this threshold are left.
5. In the last step, the neighbouring cells to the pre-clusters whose signal/noise ratio exceeds `lastNeighbourSigma` are added to the cluster. In the special case when `lastNeighbourSigma`=0, the cells are added independently of their energy.
6. In case that a neighbour is found that has already been assigned to another cluster, both clusters are merged and considered to be seeded by the sell with the highest energy.

The values used for the ALLEGRO simulations are:

- `seedSigma`=4
- `neighbourSigma`=2
- `lastNeighbourSigma`=0

The noise map used for the ALLEGRO simulations is created with the `CreateFCCeeCaloNoiseLevelMap` tool. The latter uses a `ConstNoiseTool` to implement a constant noise of approximately 2.9 MeV for the HCAL barrel cells and 1.9 MeV for the HCAL endcap cells, and a `ReadNoiseVsThetaFromFileTool` to read the noise estimated with external simulations and analytic calculations and saved in histograms of noise as a function of θ for each ECAL layer.

6.8.3 Calculation of cluster properties

6.8.3.1 Cluster energy corrections for upstream and downstream losses

By default, the energy of a calorimeter cluster is equal to the sum of the energies of its constituent cells. The small amount of energy lost upstream of the calorimeter due to interactions with the inner structures and the energy leaking beyond the calorimeter due to non full containment of the shower can be corrected for using the `CorrectCaloClusters` algorithm. The energy lost upstream (downstream) of the calorimeter is estimated from the fraction of cluster energy in the first (last) layer in the calorimeter. Parametrisations of the energy corrections as a function of these quantities with empirical analytic formulae are determined from simulated samples of single particles, making the material downstream and upstream sensitive to the passage of those particles, so that the energy deposited within is recorded during the Geant4 simulation. The `CorrectCaloClusters` algorithm takes as input a `ClusterCollection` (property `inClusters`), lists of functions (`upstreamFormulas`, `downstreamFormulas`) parametrising the dependence of the energy corrections on the cluster energy fractions in the first and last layers, and the values of the parameters of the functions (`upstreamParameters`, `downstreamParameters`). The input clusters are cloned, their energies are recalculated applying the corrections, and assigned to the new clusters, which are appended to the output `ClusterCollection` (property `outClusters`).

6.8.3.2 Cluster barycentre

The barycentre of a cluster by default is calculated as the energy-weighted three-dimensional position of its cells. Similarly, the barycentre of each layer is calculated from the energy-weighted three-dimensional position of the cells in that layer. Due to the finite granularity of the cells, their projectivity in the θ-direction, and the positioning of the cells at the geometrical centre of the cells by the digitisation algorithm, a dependence of the θ response of the calorimeter as a function of the impact point of the particle (electron or photon) on the cell is observed in the simulation, leading to an "S-shaped" response curve, consistently with the literature ("S-curve"). It has been shown that a possibility to recover a Gaussian response and improve the θ resolution is by training a multivariate regression (BDT, neural network) on simulated single particle samples that would correct for this effect. An alternative solution, implemented in Gaudi in the `k4RecCalorimeter` package, is to use logarithmic weights for the calculation of the cluster barycentre. For each layer, the weights use to calculate the cluster barycentre from the cells in the layer are calculated as $w = w_0^{\text{layer}} + \ln \frac{E_{\text{cell}}}{E_{\text{layer}}}$. The offset w_0^{layer} is chosen in order to give the best resolution in the estimated θ coordinated of the cluster barycentre in a given layer.

The calculation is performed by the `AugmentClustersFCCee` algorithm. The algorithm takes as input a `ClusterCollection` (property `inClusters`), lists of system identifiers (`systemIDs`), names (`systemNames`), corresponding readouts (`readoutNames`), number of layers for each system (`numLayers`), name of the layer field in the `cellID` encoder (`layerFieldNames`), and lists of offsets to be used for the calculation of the weights for each layer of each sub-detector (`thetaRecalcWeights`).

If an offset is negative, a linear re-weighting is used to calculate the layer barycentre. In output, a new `ClusterCollection` (property `outClusters`) is created. The output clusters are identical to the input ones, but they are augmented with `shapeParameters` (a vector of floating point quantities) containing the energy fraction, θ and ϕ of the barycentre of each layer. The names of the quantities that are stored in the `shapeParameters` vector are saved in the metadata information of the output cluster collection in the output file. In the future, the θ and ϕ coordinates of each layer could be used, together with the expected θ and ϕ resolution per layer (estimated as a function of the cluster energy), to calculate the cluster barycentre and direction, without assuming the cluster to be projective with respect to the interaction point (e.g. in the case of non-pointing photons produced by the decay of some long-lived particles).

6.8.3.3 Cluster energy regression

For the estimation of the cluster energy, a more refined alternative to the corrections for upstream and downstream energy losses consists in a multivariate regression of the cluster energy. A `Gaudi` tool (`CalibrateCaloClusters`) has been written to return an estimate of the ratio $E_{\text{true}}/E_{\text{reco}}$ of the particle that produced the cluster, using as inputs the cluster energy E_{reco} and the fraction of energy in each layer i, f_i. The regression relies on a boosted decision tree, typically trained on a sample of simulated electrons or photons with energy between 0.1 and 105 GeV. The training is performed using a small set of tools [99] that leverage the `LightGBM` [100] BDT regression classes. The trained model is saved into an output format in the portable ONNX format and can then be processed by the `CalibrateCaloClusters` algorithm. The latter takes as input a `ClusterCollection` (property `inClusters`), lists of system identifiers (`systemIDs`), corresponding readouts (`readoutNames`), number of layers for each system (`numLayers`), name of the layer field in the `cellID` encoder (`layerFieldNames`), and name of the file containing the trained model. The model is then applied over each cluster (the energy fractions are directly read from the `shapeParameters` of the clusters if they have been produced by an `AugmentClustersFCCee` algorithm, otherwise are recomputed on-the-fly). In output, a new `ClusterCollection` (property `outClusters`) is created. The output clusters are identical to the input ones, but their energies are updated with the calibrated value, and the original energies are added to the `shapeParameters` of the output clusters.

6.9 Calorimeter jet clustering

Separate tools for reconstructing jets with different input types are provided, with a more generic helper class that is used to reduce code overlap. The helper class takes as input a vector of PseudoJets corresponding to the jet inputs, and returns a vector of PseudoJets that correspond to the clustered jets. In order to enable matching between individual jet inputs and jet constituents, the helper class includes a class that derives from FastJet's UserInfoBase [44] that can be used to add an index to each jet input, corresponding to the original index of a cluster. Calorimeter cluster objects have information on the spatial position and energy, which is converted to a four-vector before passing them to the helper class. These are stored as

a `ReconstructedParticleCollection`, and the associated jet constituents are added as clusters to the jet.

Several options are provided to give flexibility to the jet definition, including the jet radius and jet algorithm. Currently, a select list of jet algorithms are enabled through a map between the algorithm name and the FastJet algorithm, but more can be included as desired. The choice between inclusive and exclusive clustering is done through the integer `isExclusiveClustering` parameter which should be set to 0/1 for inclusive/exclusive clustering. In the case of inclusive clustering, the minimum jet p_T may be specified, and in the case of exclusive clustering, the number of jets can be specified.

6.10 Truth jet clustering

Truth jet clustering is performed by a separate tool that relies on the same helper class as for calorimeter jet clustering. Particles are converted into four-vectors in order to pass them to the helper class, which returns a collection of `ReconstructedParticle`s, corresponding to the truth jets. Since these objects do not have functionality to directly associate the `MCParticle` constituents, the links between the truth particles and the truth jets are added using the `MCRecoParticleAssociation`. Each constituent adds one association to a jet, which are then written out as a collection that can be used to link jets and their constituents. The truth jet clustering tool has the same options as the calorimeter jet clustering algorithm, including the jet radius, clustering algorithm, and more.

6.11 π^0/γ identification

A multivariate classification algorithm for the discrimination between photon and π^0 candidates has been implemented in Gaudi. It relies on a certain number of input features (also called shower shapes in the following). The model is a boosted decision tree trained on simulated single particle samples of photons and neutral pions with energy between 1 and 100 GeV, split in training and validation sub-samples. The list of shower shapes contains, in addition to the cluster energy and fraction of energy in each layers, variables describing the shower transverse profile, such as the RMS along the θ and ϕ directions, and variables that try to identify the presence in the cluster of two showers induced by the two photons from π^0 decays. Examples of the latter include the difference between the second local maximum and the minimum between the first two maxima of the energy profile vs θ in a given layer, or the asymmetry (difference over sum) between the energies of the two aforementioned maxima. The training is performed using a small set of tools [101] that leverage the `LightGBM` BDT classification classes. The names of the input features and the parameters used for the training are saved to a configuration file in `JSON` format, while the parameters of the trained models are saved to an output file in `ONNX` format.

The `PhotonIDTool` algorithm takes as input a `ClusterCollection` (property `inClusters`), the name of the `JSON` configuration file and the name of the `ONNX` model file. If the metadata of the `inClusters` contains all the features needed by the classifier, then their values are read event-by-event for each input cluster, the model is

evaluated and a probability for the cluster to originate from a photon is calculated. In output, a new `ClusterCollection` (property `outClusters`) is created. The output clusters are identical to the input ones, but their `shapeParameters` are extended to contain the estimated probability of the photon hypothesis.

6.12 Pandora particle-flow algorithm

Pandora is a particle-flow algorithm developed to study particle-flow calorimetry [102]. It has been extensively studied and optimised for Silicon-Tungsten (SiW) calorimeters, particularly those developed by CALICE for linear collider experiments, over the past two decades. In the linear collider context, PandoraPFA operates through the `MARLIN` interface, `DDMarlinPandora` [92]. Like all `MARLIN` processors, `DDMarlinPandora` can also be utilised within the Gaudi framework at Key4hep via the `MarlinWrapper`, as described in Section 2. Given that `DDMarlinPandora` was originally developed for SiW calorimeters, its direct application in FCC is currently limited to the CLD detector. To explore whether PandoraPFA can be extended to other FCC detector models, a study was conducted within the Key4hep framework to generalise its usage across various detector designs. Notably, the geometry of a Liquid Argon (LAr) ECAL differs significantly from that of a SiW calorimeter. PandoraPFA reconstructs clusters based on the shower depth within the calorimeter, which is characterised by parameters such as radiation lengths, interaction lengths, detector dimensions and other material-specific properties. These properties are accessed by PandoraPFA through the `DD4hep` class `Layered Calorimeter Data`.

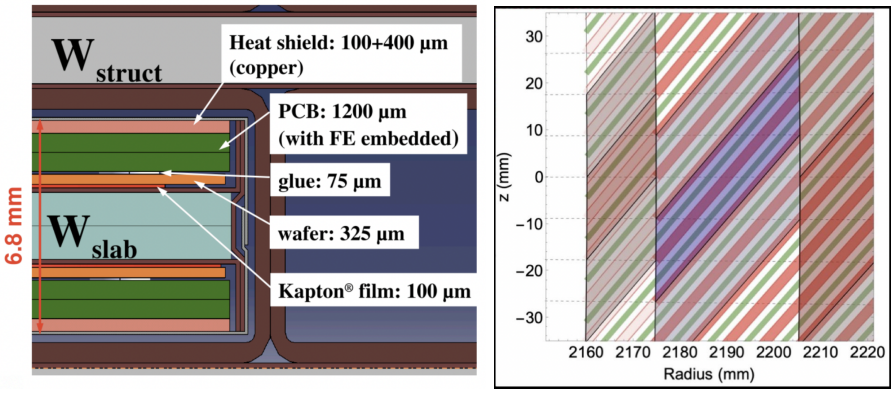


Fig. 44: (a) Cross section of a SiW ECAL slab [103]. (b) Readout cell definition of the noble liquid calorimeter.

Fig. 44a represents a SiW tile, and Fig. 44b shows the LAr detector highlighting the LAr cell. It is evident that the fundamental geometrical features of LAr and SiW calorimeters differ significantly. The LAr detector is composed of steel/Pb absorbers and readouts submerged in liquid argon, positioned at an angle of 50 ° relative to the radius (see paragraph 4.5.2.1). In a given cell of the noble liquid detector, the

number of absorbers and readouts can differ from cell to cell, as does the probability of a particle encountering an absorber or readout first. This variation impacts the material composition and properties of the calorimeter regions traversed by particles, making the determination of material properties in the LAr calorimeter more complex compared to the SiW calorimeter.

Instead of determining the material properties of individual components, the material within a given detector region is averaged, and the properties of this composite material are estimated. This process is handled by the DD4hep class, `MaterialManager`. It was established that with this method, the radiation length for the LAr-ECAL was correctly calculated to be approximately 22 X_0 as expected [104]. This method enables obtaining material properties dynamically and independent of any detector model [105]. Using this method, an interface code was created to be used between the detector and PandoraPFA which was initially implemented inside the detector geometry driver code.

6.12.1 CLD with the LAr ECAL

With the interface code enabling PandoraPFA to extract material properties from the LAr detector, a study was conducted to assess the feasibility of constructing and observing Pandora particle-flow objects in such detectors. Since Pandora requires detailed information from both `CaloHits` and `TrackHits`, a full simulation of the detector is essential. Given that the full simulation software for the ALLEGRO detector is still in development, the CLD model was used and the SiW ECAL was replaced with the larger LAr ECAL, profiting from the DD4hep plug-and-play mechanism. Given the size difference, modifications to the geometry of the outer sub-detectors was required to accommodate the LAr ECAL and avoid overlaps among subdetectors.

6.12.2 Pandora PFOs in the LAr ECAL

A set of 1000 photon events are simulated for the CLD-LAr detector at an energy of 10 GeV. During reconstruction, the `SimCalorimeterHit` collections are digitised by calorimeter digitiser as mentioned in Section 6.5. These digitised collections are further provided to PandoraPFA where Pandora can reconstruct the particle clusters based on methods given in [102]. As seen in Fig. 45, particle-flow objects (PFOs) could be reconstructed by Pandora and observed at a LAr calorimeter.

To assess the accuracy of energy reconstruction by the Pandora Particle Flow Algorithm (PandoraPFA) within the LAr calorimeter, a detailed study of the energy distribution of reconstructed Particle Flow Objects (PFOs) was performed. Figure 46a shows the energy spectrum of photon PFOs reconstructed by PandoraPFA for a 10 GeV photon test sample. The distribution demonstrates that PandoraPFA reconstructs photon energies consistently around the expected value. This indicates that the interface code between the LAr calorimeter and PandoraPFA is functioning correctly, and that the algorithm reliably reconstructs particle energies in this calorimeter environment. The interface code was also successfully employed to reconstruct particles using PandoraPFA for the ALLEGRO detector [106]. To enable detector-agnostic applicability, the interface was subsequently modularised into a plug-in. The resulting photon energy distribution obtained using this plug-in is presented in Fig. 46b.

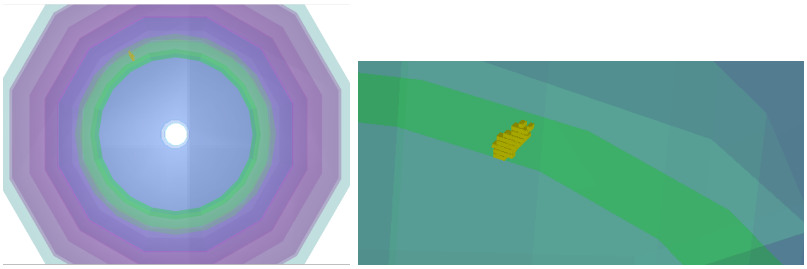


Fig. 45: (a) The event display for CLD-LAr with a photon reconstructed (yellow) in LAr (in green). (b) Close up version of the left event display.

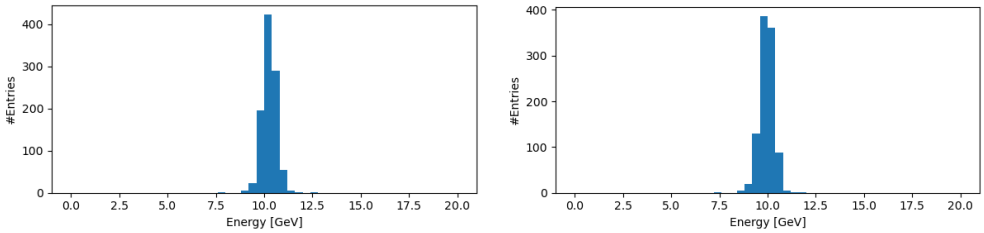


Fig. 46: (a) Distribution of the reconstructed Pandora particle-flow objects (photons). (b) Distribution of reconstructed Pandora particle-flow objects using the plug-in. Both the distributions are made using different simulated files.

6.12.3 Outlook

One challenge in this study is that, in addition to requiring calorimeter material properties and dimensions, Pandora also needs the cell sizes of the calorimeter for accurate reconstruction of PFOs. As previously mentioned, the calorimeter material properties are provided to Pandora via the `Layered Calorimeter Data` through the detector builder source code. However, the readout cell sizes cannot yet be determined, as they depend on the detector segmentation, which is defined during detector simulation. A solution to address this issue is currently under investigation.

Another caveat of the current approach is the need for data converters and wrappers to call the `MARLIN` processors into Gaudi based applications. To improve the framework homogeneity and robustness, `DDMarlinPandora` is being ported into a native Key4hep algorithm. This nearly completed effort will enable us to eliminate the cumbersome wrappers and converters.

6.13 Machine learning based particle-flow

ML-based particle-flow is under development and is currently performed by a separate repository external to Key4hep. The algorithm has been tested with data using the

CLD detector but it could easily be adapted to data from other detectors. The inputs to the algorithm for CLD are:

- ECAL calorimeter hits (collection `ECALBarrel`)
- HCAL calorimeter hits (collection `HCALBarrel`)
- Muon system hits (collection `MUON`)
- Reconstructed tracks (collection `SiTracks_Refitted`)

The algorithm does not take as inputs EDM4hep files directly but a flat root tree that is built from the EDM4hep files, we refer to this format as an ML-ready data format. This format allows to only store relevant information and to preprocess information about the hit to particle links. A framework exist to convert EDM4hep files to the ML-ready format, and in the future the algorithm will be implemented as a Gaudi algorithm.

The output of the algorithm is a collection of particle-flow candidates with an assigned 4-vector and PID.

The method behind this algorithm is a geometric graph neural network approach. The algorithm processes each input hit into a geometric algebra representation and uses a pre-trained graph neural network architecture and a clustering algorithm (HDBSCAN [107]) to obtain a label for each hit/track. The different labels result in the set of reconstructed particles (which comprise hits and tracks). The output of this clustering algorithm is fed to another graph neural network which determines the properties of the reconstructed particles. The details of the algorithm are available in a dedicated note [108]. For neutral particles, defined with clusters where no track is present, the properties are obtained as follows

- PID: predicted from concatenated *high-level features* and the outputs from a GATr head using all hits and track assigned to the particle during the clustering step. The result is a score for each class (e^-, γ, muon, charged hadrons and neutral hadrons). The definitions of the *high-level features* are available in the note [108].
- Energy correction: the energy correction is estimated by also concatenating high-level features with the outputs of a GATr head. The concatenated vector is passed through an MLP which gives the correction factor to the energy.
- Direction estimation: similarly to Pandora, we estimate $\frac{\vec{p}}{|p|} \sim \frac{\vec{w}}{|w|}$ where $w = \frac{\sum_i E_i \vec{r_i}}{\sum_i E_i}$ is a vector pointing from the interaction point $(0, 0, 0)$ to the energy-weighted average of the hit coordinates $\vec{r_i}$.

For the charged reconstructed particles:

- PID: estimated in the same way.
- Energy correction: obtained from the track momentum.
- Direction estimation: is obtained as the direction the normalised track momentum from the track state at IP.

The particle mass is estimated from the PID classification. For the charged hadrons, the mass is set to m_π and for the neutral hadrons to m_n.

Different evaluation metrics are also available to compare the performance against PandoraPFA:

- Energy resolution per particle type
- Angular resolution per particle type
- Fake rate and fake energy rate per particle type
- Particle reconstruction efficiency

The definition of the metrics and details about the training can also be found in the associated note [108].

6.14 Outlook

Though the digitisers currently available in the FCC Software are already enabling many interesting studies by providing a parametrisation of sub-detectors hit spatial resolution and efficiency, some effects such as the detector performance degradation due to beam induced background are better estimated with a more detailed treatment of the detector responses. Modelling a higher level of detail in the digitisers will also enable the optimisation of the detector electronics based on full simulation. Here is a non-exhaustive list of ongoing or planned implementations:

- Planar tracker digitisation with charge sharing, hit clustering and energy weighted position assignment with different assumptions on the number of bit available to encode the energy deposit per sensitive element
- Noble liquid calorimeter digitisation with a detailed signal generation taking into account the effect of widening sensitive gaps resulting in a non-constant electric field, an emulation of the amplifier and shaper and a peak finding or optimal filter implementation.
- Drift chamber digitisation with waveform construction (e.g. based on Garfield integration into Geant4 or a Garfield based paramterisation, possibly using machine learning) and an emulation of the front-end electronics response including cluster counting.

Most of the work ahead lies though in the implementation of higher-level reconstruction algorithms. Indeed, while significant progress has been made, the currently available solutions do not cover all the needs yet, especially, but not only, regarding particle identification.

As stated earlier in this note, the digitisation and reconstruction tools should remain as generic as possible in their implementation to cope with the ever evolving nature of the FCC detector designs. While this is relatively easy for simple algorithms, it will be an interesting challenge to implement more advanced solutions with minimal detector specific components.

7 Analysis tools

The outcome of a typical physics performance analysis for the FCC feasibility study is often a set of histograms or an unbinned collection of events, which are subjected to statistical analysis to extract the desired physics quantities, whether or not under different detector conditions. This final step should typically be performed quickly, which can be achieved by skimming and slimming the event collection, as well as by utilizing modern computational tools to expedite the process of looping over event samples. The analysis tools should be flexible, fast, easily extendable, and hosted within an ecosystem that encompasses:

1. A convenient framework for processing events, skimming/slimming them, and producing final histograms;
2. Interaction with a database of centrally produced Monte Carlo samples;
3. Tools for plotting, statistical analysis and event display;
4. Various tools and code snippets that are useful for common analyses;
5. Ability to analyse both fast and full simulation event samples;
6. A central repository for analyses, which can be used to version the code and serve as examples for future work.
7. Optional functionality to alter detector parameters on the fly (fast simulation).

Moreover, the entire analysis framework should serve as an educational platform with easy on-boarding as the majority of the new students and users is coming from the currently running big experiments at CERN (ATLAS, CMS, ALICE, ...), allowing them to perform simple analyses without requiring detailed background knowledge in physics or programming. Several examples are provided to assist in conducting thorough analyses of various types. The main analysis framework used for the FCC feasibility studies, *FCCAnalyses* [109], is based on RDataFrame [110] and is fully embedded in the Key4hep software stack. The code for the studies is collected in two repositories, which act as a catalogue of sorts: FCCeePhysicsPerformance [111] for the FCC-ee studies and FCChhPhysicsPerformance [112] for FCC-hh studies.

An overview of FCCAnalyses capabilities is given in Section 7.1. Other available analysis solutions are presented in Section 7.2. Auxiliary database of the centrally produced samples is discussed in Section 7.3. This chapter concludes with a discussion of future improvements and directions for analysis frameworks in Section 7.4.

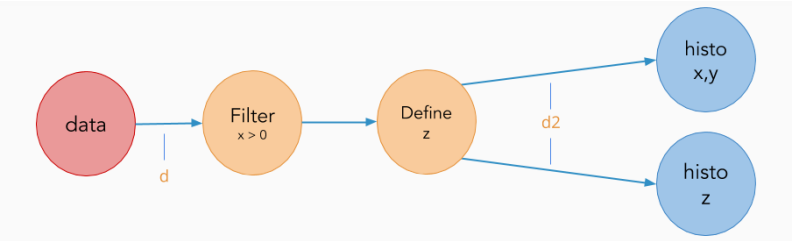
7.1 FCCAnalyses framework

The ROOT's RDataFrame provides a very efficient execution engine, however an analysis framework requires few more components, which are usually dependent on the software ecosystem of the particular experiment. That is why the FCCAnalyses framework [109] offers, on top of the ROOT RDataFrame itself also the management of the input samples metadata (for centrally or locally produced samples), standard set of functions to be used in the actions and transformations of the dataframe, ability to run on distributed system — HTCondor, and the possibility to arrange the analysis into multiple stages.

7.1.1 ROOT RDataFrame

With the introduction of RDataFrame [110], ROOT [10] offers a convenient high level abstraction inspired by functional programming to describe an analysis. The analysis is expressed as a chain of actions and transformations which at the end produce physics output (histograms, graphs, ...). The execution of the chain itself is done independently of the user in an optimised way. The optimisations in place are caching, lazy evaluation and parallel execution (multithreading).

Users define functions for the actions and transformations in C++, however the design of the analysis chain itself can be done in equivalent way either in Python or C++. An example analysis graph and corresponding code can be seen in Fig. 47.




```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z","x*x + y*y");
// make histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D({"hxy","hxy",16,-1,1,64,-1,1},"x","y");
```

Fig. 47: Graph of the simple analysis chain composing of a filter and a definition of a new variable resulting in two histograms.

7.1.2 Analysers

Collectively, the set of all analyser functions and functors is called analysers. To write an analysis, users need to employ them to create new columns in the dataframe. The analysers can greatly vary in complexity and the users are encouraged to upstream their analysers to a central repository.

An example of an analyser is shown below. This analyser takes as input a column containing reconstructed particle and a column containing associations between reconstructed particles and Monte Carlo particles. The output of the function is the Monte Carlo particle associated with the input reconstructed particle.

In order to provide fully fledged EDM4hep collections and objects to the users writing their own analysers, a specialised ROOT RDataSource [113] API has been implemented in PODIO [12]. It acts as an intermediate layer between the ROOT file and the RDataFrame, which constructs EDM4hep objects and relations between them.

7.1.3 Running modes

Depending on the analysis, the workflow for obtaining the necessary histograms can vary. For simple analyses, tasks can be performed iteratively, allowing histograms with final or partial results to be generated by looping over events in the EDM4hep files. This iterative approach is intuitive for inexperienced analysis writers, facilitating their engagement with the analysis process. For more complex analyses that involve intensive and time-consuming calculations (e.g. flavour tagging and inference), the workflow needs to be optimised by either skimming the datasets or producing a secondary or tertiary set of files containing the desired variables, which can then be used for event selection and histogram generation.

Both running modes are supported by FCCAnalyses and are referred to as *histmaker* and *staged* running modes. The former directly produces the desired histograms by analysing the input EDM4hep files, while the latter generates intermediate ntuples (ROOT trees) that are in a final stage. The *staged* running mode is compatible with batch systems (e.g. HTCondor or Slurm), making it convenient for processing larger datasets.

The following examples illustrate the usage of both running modes:

- As an example of the analysis making use of the *histmaker* workflow, the derivation of the Higgs boson mass [114] based on recoil quantities requires a tight event selection of two leptons to effectively reject large backgrounds. Since it does not involve heavy computations, it can efficiently be run in a single step, which produces all needed histograms. The recoil mass distribution is then parametrised analytically using RooFit in scripts not dependent on FCCAnalyses and injected into the CMS Combine tool to perform a statistical fit, ultimately extracting the uncertainty on the Higgs mass.
- The *staged* running mode is on the other hand better suited for the analysis measuring the hadronic Higgs couplings, which requires extensive use of jet clustering and flavour tagging [115]. Due to the computational expense of flavour tagging inference, the first step in the *staged* mode of FCCAnalyses involves creating ntuples that contain the basic event selection, jet definitions, and per-jet flavour tagging probabilities. This is followed by a fast final step that completes the event selection and categorises events, assisted by a multivariate classifier. The output is a multidimensional histogram for each probed final state, which is then fed into Combine to perform a multidimensional fit and extract the uncertainties on the hadronic Higgs couplings.

7.1.4 Integrated tools and add-ons

Several tools and add-ons are integrated into FCCAnalyses to facilitate the use of dedicated physics software packages for the users. A list of the implemented tools is provided below.

Function library. FCCAnalyses features both lower-level and higher- level function catalogues, comprising C++ snippets commonly used across various analyses. Higher-level functions facilitate interaction with objects in the collections, such as selecting specific particles or identifying the associated Monte Carlo particle from

a reconstructed particle. Lower-level functions are utilities for physics calculations, including conversions to Lorentz vectors, invariant mass calculations, and recoil calculations.

FastJet interface. FastJet is an external software package widely used in high-energy physics for jet finding, the process of identifying and clustering hadronic jets [44]. It offers efficient clustering algorithms tailored for electron-positron and hadron colliders, along with tools for analysing jet properties. FCCAnalyses is compiled with FastJet by default and provides an interface for performing jet clustering directly from the reconstructed particle collection. Additional functions, such as matching jets to the original particle collection and identifying the true flavour of a jet by matching it with the closest truth particle, are also implemented.

Flavor tagging. ParticleNet is a deep learning-based model utilizing graph neural networks for jet tagging in high-energy physics, effectively classifying jets by analysing the relationships between their constituent particles [40, 116]. Training has been performed on electron-positron events, primarily using events from Higgs decays to hadrons, and is integrated with FCCAnalyses. Inference is applied using ONNX, which has been made thread-safe for execution. Since flavour tagging is closely linked to jet clustering, Python helper classes have been written to perform both tasks in a single dataframe function.

Support for multivariate analysis techniques. In addition to ONNX integration, which supports any type of inference, dedicated functions have been implemented to perform tasks such as BDT (Boosted Decision Trees) using the TMVA or XGBoost [117] packages. Several analysis examples are available, explaining both the training and application steps.

Smearing utilities. For the FCC feasibility study, it is crucial to systematically assess the impact of detector performance on physics output. To avoid the need for re-generating simulation samples with Delphes for different detector configurations, a set of smearing tools has been implemented. These tools modify the performance of specific detector components and recompute certain event-level quantities. The implemented features include covariance matrix smearing (to model tracking performance), energy and momentum smearing for specific particle types, and smearing of the interaction point (to account for flavour performance).

Plotting tools. Histograms can be directly plotted using a custom plotting library in FCCAnalyses built on ROOT. Several options are available for plotting, such as stacked histograms or background vs. signal comparisons, with proper normalisation based on the number of processed events, cross-section, and integrated luminosity.

Yield tables. Upon user definition of a set of cuts, the framework enables automatic generation of yield tables, incorporating cut efficiencies, in both LaTeX and JSON formats. This yield table generation tooling can also be utilised to produce cut flows, provided the set of cuts are designed in chained fashion.

Statistical analysis. Histograms are used as inputs to auxiliary tools for performing statistical analysis and extracting the desired physical quantities along with their associated uncertainties. A common tool for this is CMS Combine [118], which uses histograms in ROOT format and a text file to encode the likelihood, including signal and background processes and systematic uncertainties. FCCAnalyses can directly

produce both the ROOT files and text datacards, which can be used directly in the Combine tool (to be installed separately).

7.1.5 Distribution of FCCAnalyses

The FCCAnalyses framework [109] is distributed in its compiled form as part of the Key4hep [6] stack, while the package definition itself lives in `key4hep/key4hep-spack` [22] Spack repository. Users are encouraged to use this already pre-compiled version of the framework and add their specific functions in the auxiliary C++ header which is compiled with the help of internal ROOT Cling compiler.

7.2 Other analysis tools

Apart from the FCCAnalyses framework, several other frameworks are available for processing EDM4hep files. These include Coffea, a Python-based framework known for its scalability and efficient handling of large datasets, and Julia- based tools, which offer high performance through just-in-time compilation. All of these frameworks share the common requirement of being able to read the evolving EDM4hep schema. In addition to full-featured frameworks, simpler event loopers in Python or C++ can be easily implemented for specific tasks, such as debugging or lightweight analysis, providing flexibility for users who need minimal overhead for quick development.

7.2.1 Julia

Julia [119] is a high-performance, dynamic programming language designed for numerical and scientific computing, renowned for its speed, ease of use, and expressive syntax. It has gained significant traction, particularly within the scientific and data science communities. Leveraging these strengths, the EDM4hep data model has been implemented in Julia to read and analyse existing ROOT file samples using the UnROOT.jl ROOT reader. A small Julia script, generated from the YAML file defining the EDM4hep schema, enables the creation of the Julia structures, allowing for complete analyses within Julia, including outputting histograms for further post-processing. Multithreading support is included, and benchmarks indicate a similar performance to RDataFrame-based analysis. While many code snippets available in FCCAnalyses have yet to be ported to Julia, the FastJet package has been successfully transitioned from C++ to Julia, achieving excellent performance.

7.2.2 Coffea

Coffea [120] is a Python-based framework designed for scalable, fast, and efficient processing of large-scale high-energy physics datasets. It enables the rapid development of analyses with minimal overhead, offering tools for histogramming, parallel execution, and applying corrections or calibrations. Coffea excels in handling data both locally and on distributed systems (such as HPC or cloud platforms) through backends like Apache Spark and Dask. Additionally, Coffea supports EDM4hep by reading the EDM4hep schema YAML files, and provides full integration with FastJet using

Awkward arrays for vectorized jet finding. It also features a user-friendly interface for various machine learning frameworks.

7.3 Database of Centrally Produced Samples

Large simulated samples, which are of interest to multiple case studies or which are unpractical to generate by an analyser on its own are produced centrally using either of two production systems, internal EventProducer or ILCDirac based FCCDirac. The production of a particular sample is carried out by a dedicated producer of the samples, who makes sure that the samples are properly generated and stored. After the generation is complete the samples are validated and published in the web database.

The web database is located at `https://fcc-physics-events.web.cern.ch`, it is freely accessible without any logins. The individual samples are organised in the following categories:

- *Accelerator*: FCC-ee or FCC-hh.
- *Type of the events*: MonteCarlo simulation, fast Delphes simulation and full Geant4 simulation
- *Generator level file type* or *Detector*: STDHEP, Les Houches or CLD, IDEA, etc.
- *Campaign*: `spring2021`, `winter2023`, etc.

Users need to select one of the options in every category to see the listing of available samples. The example listing of samples available in `devel` campaign of Geant4 full simulation for FCC-ee is shown in Fig. 48.

7.4 Outlook

As the analysis strategies of the following physics studies for the pre-TDR phase are expected to grow in complexity, the analysis ecosystem of the FCC will need to undergo an overhaul. The event processing engine of RDataFrame has proven it's ability to quickly process large amount of events in parallel fashion. Its further improvements and capabilities will be closely followed and discussed with the ROOT team. The focus of the FCCAnalyses framework will be on the following:

- Enhance the integration of machine learning tools
- Portability to other facilities outside CERN
- Support for wider range of distributed computing platforms
- Investigating flattening/optimizing of the current data model
- Enhance interoperability with other HEP analysis tools
- Extend plotting facilities
- Implement a sample management system which is independent of FCCAnalyses constrains

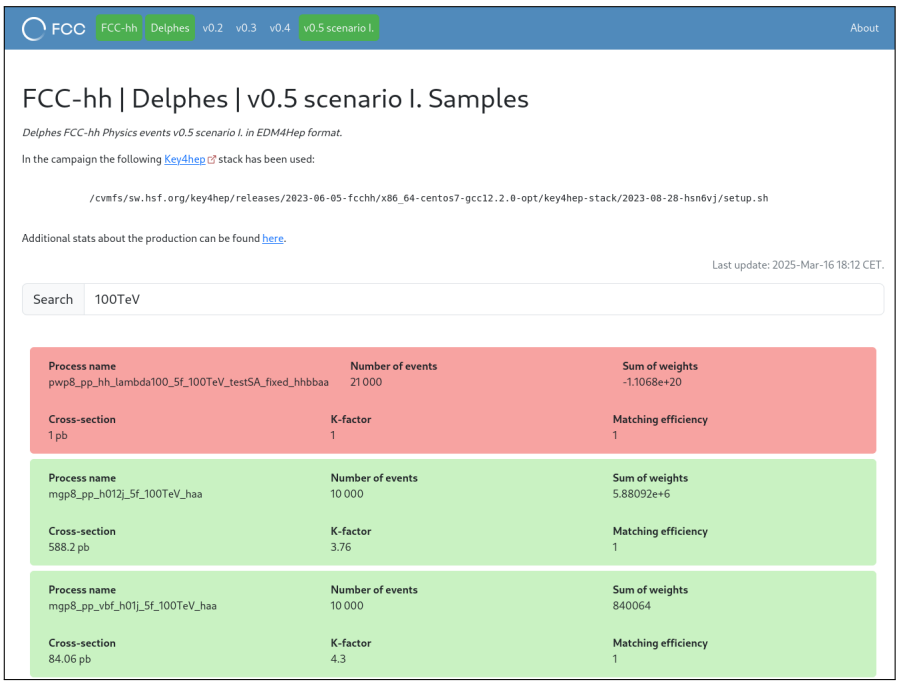


Fig. 48: Example listing from the FCC database of centrally produced samples. The listing shows FCC-hh Delphes samples from `v05` campaign using scenario I detector.

8 Visualisation

The role of visualisation tools at FCC is twofold: on the one hand, they help in the understanding of the particular task (e.g. detector design, simulation, analysis), and on the other hand, they help to convey messages to a larger audience (e.g. presentations, outreach). Since there is usually overlap in visualisation needs between event reconstruction and detector design, the prevailing approach is to provide a precise depiction of the detector onto which event data are overlaid. Another approach, which is more suitable for analysis and outreach, is to simplify or discard the visualisation of the detector and focus on visualising the event content itself.

Since FCC Software [121] is based on Key4hep citekey4hep, there are many solutions available that have been developed over the years, ranging from tailored, one-purpose solutions (e.g. to visualise hits in the calorimeter) to general-purpose ones shipped with a large framework (e.g. JSROOT [122]). An example of visualisation of the CLD [54] detector in JSROOT can be seen in Fig. 49.

8.1 Detector visualisation

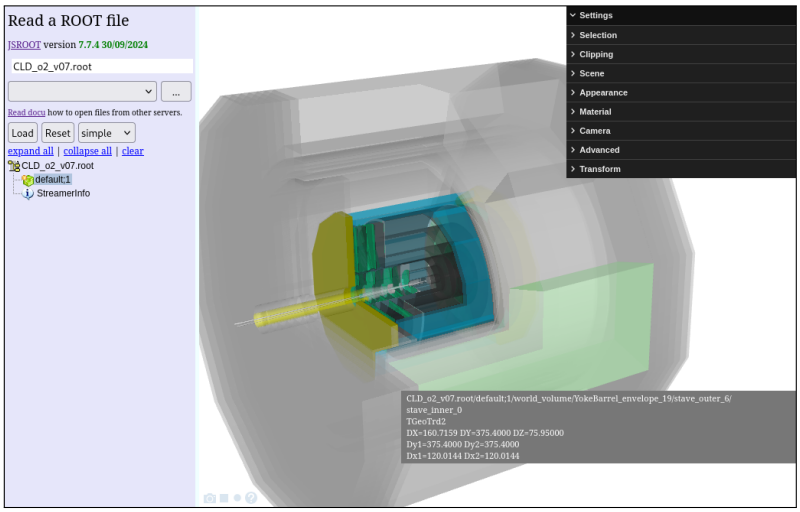


Fig. 49: Visualisation of the CLD [54] detector in JSROOT [122].

Among the tools that can be used for detailed study of the detector are:

- Geant4 Qt visualisation — This solution creates scene with detector geometry, particle trajectories, etc, and features clipping, volume rendering, file export, and more.
- JSROOT — The primary visualisation solution for ROOT [10] since version 6.28. It is capable of visualising histograms, graphs, geometry objects, and more.
- CED [123] — Ported from iLCSoft and supported in Key4hep, this solution was developed for the detectors of the linear collider concepts.

- geoDisplay and DDeve — Both are part of DD4hep [1] project. DDEve is based on EVE [124], ROOT's event visualisation environment.

Overlay of event data is supported in almost all of the solutions listed above. The list of commonly implemented features includes the ability to select only a subsection of the geometry hierarchy tree for visualisation, clip along single or multiple planes, adjust visibility levels, and manipulate transparency and colour. Detector descriptions can typically be provided in a DD4hep compact file, a ROOT file, or a GDML [125] file.

Purpose-built solutions developed in the context of software development targeting the FCC include `calodisplay` [77], a tool designed to help understand the simulation of calorimeters within the ALLEGRO detector concept (see Fig. 26) but that could be generalised.

8.2 Event visualisation

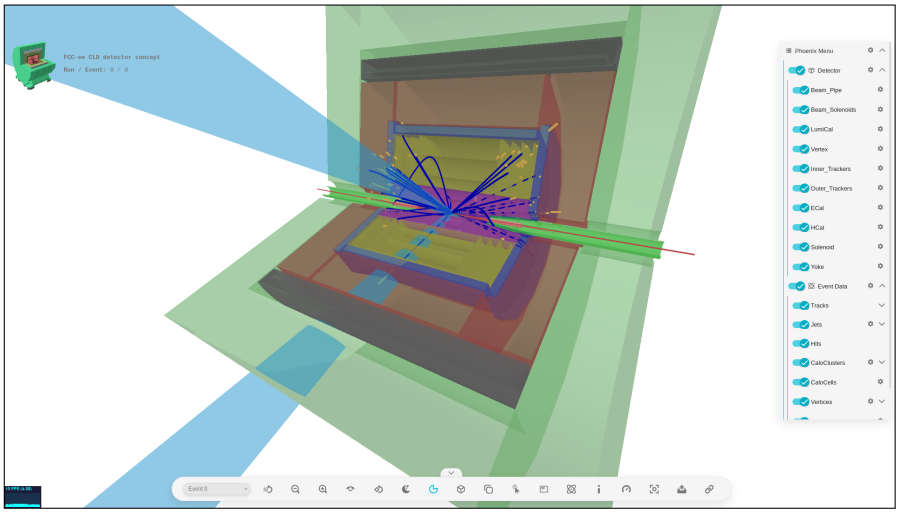


Fig. 50: Visualisation of a $t\bar{t}$ event inside the CLD detector concept in Phoenix.

In the case of FCC-ee, the events are significantly cleaner compared to the proton-proton collisions at the LHC. This means that it is feasible to fully visualise all objects in the event without extensive clean-up.

In event display solutions, the detector geometry is usually simplified and serves primarily as a background. The event data are then visualised using geometric shapes that represent particular types of event data. Tracker or calorimeter hits are shown by highlighting individual simplified cell representations; tracks are visualised as 3D lines; calorimeter towers are visualised as bars, with their height dependent on the tower's energy; jets are visualised as cones; and missing E_{T} is visualised as a dashed line.

One of the popular detector-independent 3D event display frameworks, Phoenix [126], was enhanced to handle event data in the EDM4hep format and is used to publish all versions of the FCC detector concepts at https://fccsw.web.cern.ch/fccsw/phoenix. Although all detector versions are published centrally, users can upload their own detector geometry as a ROOT or glTF [127] file. The workflow for visualising an event begins with preparing the event data: converting it from ROOT binary format into JSON format, opening the page with the desired detector concept, and finally loading the event data onto the website. The website is static, meaning all user data (event data or detector geometry) remains on the user's local machine. An example of 3D event visualisation using Phoenix is shown in Fig. 50.

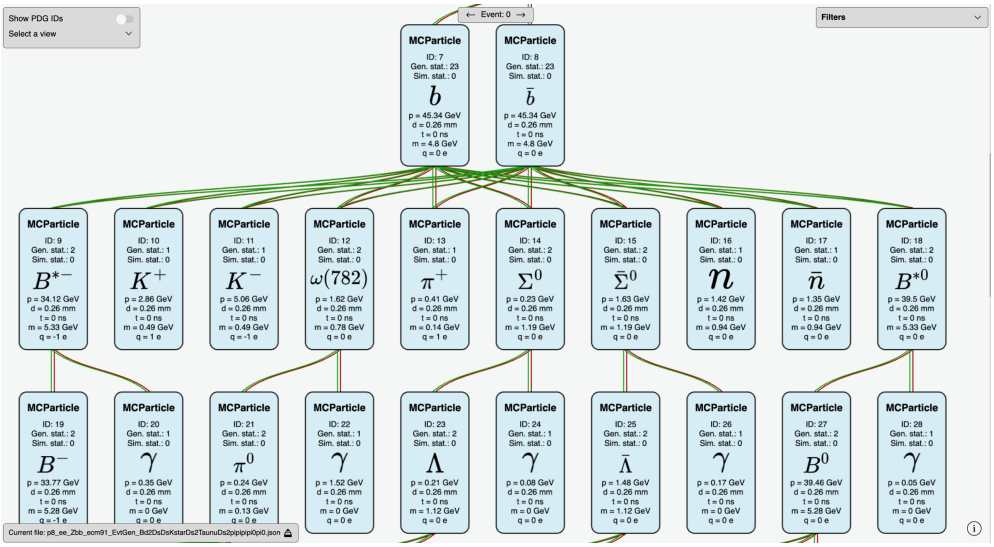


Fig. 51: Visualisation of the Monte Carlo particle tree for an $e^+e^- \to Z \to b\bar{b}$ event, generated using the EDM4hep Event Data Explorer [128].

8.2.1 EDM4hep event data explorer

While 3D-based event visualisation is useful for understanding events, it is usually unable to accommodate all the data required for thorough debugging. One solution that allows for comprehensive event data investigation is called Eede [128], for 'EDM4hep event data explorer'. It is a web application that aims to offer a detailed inspection of events according to the EDM4hep Data Model. It currently includes the ability to visualise a subset of the most important collections. Currently supported are `MCParticle`, `ReconstructedParticle`, `Cluster`, `ParticleID`, `Track`, and `Vertex` collections. It can also represent *relationships* (inherent structure of the collection) and *links* (map between reconstruction and simulation objects).

As shown in Fig. 51, Eede draws multiple *info boxes* where each one represents a particle from a Monte Carlo particle tree, including data about a subgroup of their properties like charge or PDG. These particles have *relationships* between them, so either a red line (parent relationship) or a green line (daughter relationship) connects two info boxes to represent these relationships. The combination of info boxes and relationship lines is known as a *view*, where a *view* is a way to represent either one, or multiple collections in a single graph, with their corresponding relations or links as determined by EDM4hep. In Fig. 51, the view is called 'MC Particle Tree', and views are developed conveniently to show as much information possible with their structure without losing order neither overloading the user.

8.2.1.1 Features

Eede has one panel at the top, two at each side, and one at the bottom left corner to provide various functionalities to help inspect events, as illustrated in Fig. 52.

- **View selector:** change between different views. Each view may contain a single collection, or multiple ones to represent associations.
- **Event switcher:** choose any event from the same file.
- **Filters:** select a specific set of particles from the same collection of the event according to charge, mass, energy, position, and other properties depending on the type of collection selected. It can also choose "sub-collections" of a collection.
- **File loader**: Allows to load another EDM4hep JSON file, replacing the currently loaded one.
- **Hover information:** when inspecting objects from a collection in detail, it shows a box with more information.

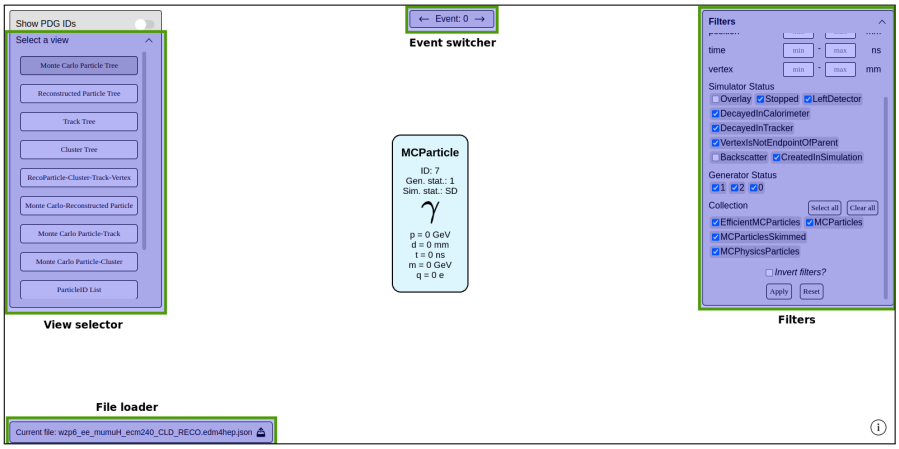


Fig. 52: Image describing Eede features showing only a single particle in a Monte Carlo Particle view.

8.2.1.2 Tech stack

Eede is a client-side website made with HTML5, CSS3, Javascript and `Pixi.js` [129]. `Pixi.js` is an open source library that allows for fast rendering of any kind of graphics using WebGL. Eede currently uses `Pixi.js` as a graphic engine because its performance is able to graph thousands of objects at the same time without overloading the CPU by applying techniques known as culling [130] which ignores elements that aren't visible in the view port, permits optimisations like lazy loading assets as text or images to consume less RAM, the community around Pixi.js is active (so it will continue to have updates), and the API is convenient to use.

8.2.1.3 Workflow

The basic workflow to use Eede consists of the following steps:

1. Convert EDM4hep ROOT file into a EDM4hep JSON file.
2. (Optional) Transfer the EDM4hep JSON to the local machine.
3. In Eede website open file picker.
4. Select the desired EDM4hep JSON file.
5. Choose a view to use to inspect the data.
6. (Optional) Select an event number. Eede selects the first event available by default.
7. Click on visualise.

Then, users are able to explore all objects and relations from the view, change to a different view, open the information menu to learn about the current view, apply some filters to the objects shown, load a different EDM4hep file or change event.

8.2.1.4 Future goals

The future goal of Eede is to be able to visualise all the collections from EDM4hep, independent of the version from EDM4hep. Based on the view, the visualisation would use an algorithm to auto graph the collection objects without overlapping a long relationship lines in order for both relations and links between collection objects to be easily understandable.

8.3 Analysis visualisation

Modern approaches to data analysis construct analysis code as a graph or a series of small, composable functions. Visualisation of these functions can subsequently aid in understanding the analysis. An example of an FCC analysis computational graph is shown in Fig. 53.

8.4 Outlook

Over the past several decades, the stability of the graphical user interface could be considered guaranteed when the visualisation task is confined to a single operating system (OS) and a single machine. Once the visualisation task crosses either of these boundaries, long-term support requires significantly more maintenance. Although there are

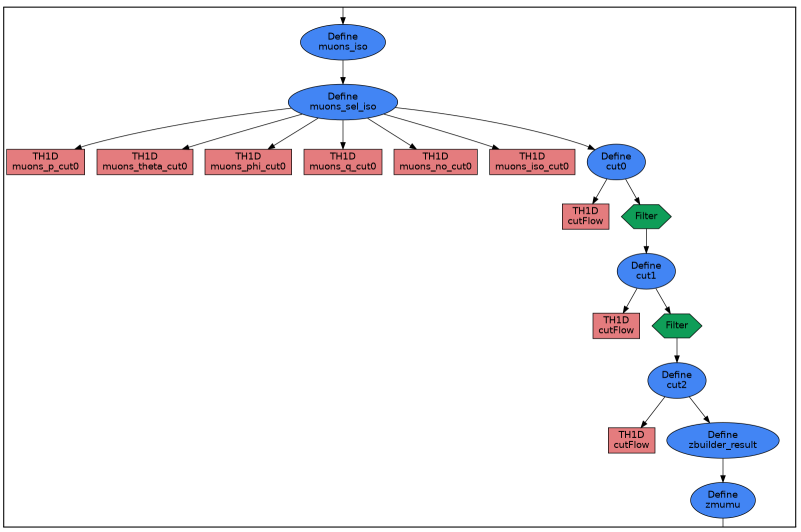


Fig. 53: Visualisation of the analysis graph using ROOT RDataFrame, showing part of FCC-ee Higgs mass recoil analysis.

many widely used solutions to address the problems that arise, it is becoming clear that web-based visualisation offers several advantages over other solutions: graphics APIs are standardised across browsers and independent of the underlying OS, updates are easily distributed, and the combination of presentation and task use-cases is straightforward.

With further development of web graphics APIs like WebGPU [131] and new approaches to executing code in the browser, such as WebAssembly [132], the performance of visualisation toolkits is expected to become more comparable to that of native, OS-dependent solutions.

The challenge for web-based solutions is managing dependency on JavaScript frameworks, as these tend to change drastically over time.

9 Distributed computing and resources

The currently available computing resources, along with long-term estimates of future needs, are detailed in the main FCC PED Feasibility Study Report. This section, therefore, focuses on the tools available for distributed computing.

9.1 EventProducer

Most of the MC events used for the various FCC physics performance studies are produced centrally using an in-house solution, the EventProducer event production system [133]. This package handles the entire event production chain from the event generation through standard MC generators to their reconstruction with Delphes. It also provides bookkeeping functionalities for the produced simulated samples, that are used to automatically fill the web database described in Section 7.3.

The EventProducer supports MC event production with the generators KKMC [134, 135], MadGraph [136] and Whizard [137, 138]+Pythia 6 [139]. The generator cards to be used for central MC production are found in the FCC-config repository [140], where various branches are available, corresponding to different, past and present production campains. In the EventProducer itself, accelerator-specific (FCC-ee or FCC-hh) configuration files contain information for all the available cards, giving information about the processes involved and their cross sections. This file and the FCC-config repository content are updated regularly to include any new process of interest for the feasibility studies. Depending on the generator choice, it produces files either in the STDHEP (Whizard+Pythia 6) or LHE (KKMC, MadGraph) format. In the cases where LHE files are generated, the EventProducer then allows one to hadronise events with Pythia 8 [47]. The latter is also used in a standalone way, typically for the generation of high-statistics, low-particle-multiplicity samples. Finally, the reconstruction is handled through the same pipeline, taking as input any STDHEP or LHE files, and one of the Delphes detector configuration cards discussed in Section 3. The outputs are ROOT files in the EDM4hep format, and centrally stored on the FCC EOS space for easy access by analysers. Full compatibility with the FCC-Analysis framework is ensured through the automatic generation of campaign and detector specific configuration files.

Users of the EventProducer typically interact with only two files in the framework: a configuration file, `param_FCC*.py`, and an executable, `run.py`. The configuration file contains information on the location of all expected input and outputs required to the event production on CERN's HTCondor batch system, as well as on the available Delphes detector configurations and processes that can be generated. For the latter, it comprises the processes names, short textual descriptions, and relevant numbers for their analysis, such as their cross sections, eventual k-factors or matching efficiencies. Every possible action is called through the `run.py` executable, itself calling the relevant modules according to the selected input arguments in a transparent way for the user. For the purpose of event generation and reconstruction, a typical call of the executable looks as follows:

```
python bin/run.py -p ${process_name} \
                  [--FCCee / --FCChh] \
```

```
[--LHE / --STDHEP / --reco] \
${generation_arguments} \
${job_type_arguments} \
${submission_arguments}
```

The `process_name` argument must correspond to the name of a generator card available in the FCC-config repository, and included in the configuration file for the corresponding collision type (`--FCCee` or `--FCChh`). Depending on the type of events to generate (`--reco`, `--LHE` or `--STDHEP`), the choice of `generation_arguments` varies, and is tailored to each supported MC event generator. The `job_type_arguments` allow one to select the action to be performed. The `--send` option is to be used for event generation and reconstruction purposes. Other options are available, but are typically not used by standard users. For the most part, these are only used by the automatic bookkeeping pipelines. Finally, as the name suggests, the `submission_arguments` allow one to handle the job submission itself. The user can chose to run locally (`--local`), or to use, e.g. CERN's condor batch system (`--condor`) for larger scale production. A more thorough description of the above options, as well as a variety of examples can be found in the `README` of EventProducer [133].

9.2 iLCDirac for FCC

iLCDirac is an extension to the DIRAC distributed computing framework. DIRAC was originally developed by and for the LHCb collaboration, and later generalised to be used by small or large communities in instances that support one or more Virtual Organisations. DIRAC provides tools and utilities to efficiently use any distributed computing resource via a central interface. In particular, DIRAC hides the complexity of the distributed infrastructure from the end-user through a uniform interface with multiple possible back-ends.

9.2.1 General description

The DIRAC workload management system provides access to computing resources. It is possible to run jobs on a laptop, for debugging purposes, batch-farms that are behind a so called grid computing element (mostly HTCondor or ARC), cloud services or high performance computing systems. In most cases the computing software will present a uniform environment to the eventual payload job by installing a "pilot", which sets up the environment and then pulls a job from the central instance. The pilot system enables a high efficiency for end user jobs, for example through ensuring a healthy worker node is available with sufficient disk space.

The data management system (DMS) of DIRAC provides access to storage elements of all variants and protocols that exist. The design of the DMS, or rather the resource abstraction, enables quick integration of new storage back-ends without compromising the user experience. The data management system also contains a meta data file catalogue, that stores information about which files exists as so called "Logical File Names (LFNs)", at which storage elements the files are located, the "replicas' of the LFNs, and allows one to add arbitrary meta data, such as event types, number of events, or related software packages. Finally the DMS contains the functionality to transfer files between storage elements, users, and computing elements.

```python
from ILCDIRAC import UserJob, FileCatalog, DDSim, DiracILC
# in principle ProdID uniquely identifies this sample already
inputData = FileCatalog().findFilesByMetadata({'ProdID': 16560,
    'EvtType': 'Hbb', 'DataType': 'stdhep'})
job = UserJob()
job.setSplitInputData(inputData[:100], numberOfFilesPerJob=1)
ddsim = DDSim()
ddsim.setVersion("key4hep_240412")
ddsim.setDetectorModel("CLD_o2_v06")
ddsim.setNumberOfEvents(1000)
ddsim.setSteeringFile("cld_steer.py")
# jobIndex will be injected in the name
ddsim.setOutputFile("ddsimout.root")
job.append(ddsim)
job.submit(DiracILC())
```

Listing 2: Example iLCDirac submission script for users. Import statements have been simplified for brevity.

Another important piece of DIRAC is the transformation system, which enables a great deal of automation for the workload and data management systems. This transformation system allows one to define concrete tasks, such as the simulation and reconstruction of a large number of events. Once the tasks are defined, the transformation system takes care of job submission, ensuring eventually reaching the desired number of events. For data management, it can ensure the consistent transfer of files from different storage elements.

All three systems – workload management, data management, transformation – rely on dedicated databases that persist information, services that allow users and other pieces of DIRAC access to the database in the form of an API and remote procedure calls, and agents that perform actions on a regular basis, from every 30 seconds, to once a day. For example the file catalogue is a service in the *DataManagement* system that provides an interface to the *FileCatalogDB*. This file catalogue service is used by the Transformation *InputDataAgent* to obtain new files to be treated by the transformations that are stored in the TransformationDB. A more detailed description of the architecture of the various DIRAC systems is part of the DIRAC documentation [141].

The final piece of DIRAC that will be described here is the WebApp, which provides a graphical user interface to monitor jobs, transformations, resources and users, for example.

The iLCDirac extension makes use of all the systems described above, and some more functionality that is described elsewhere[142–144]. The iLCDirac extension itself provides interfaces to the commonly used software of the ILC, CLIC, and FCC communities. This provides a convenient python API for users to describe most common tasks they will encounter, and iLCDirac will ensure that the input files will be available, the software installed, and the output files placed where users can use them later. Listing 2 shows the python script that will create 100 simulation tasks of $H \to \overline{b}b$ events using the CLD detector model.

9.2.2 Workflow modules

The main developments to make iLCDirac usable as the FCC distributed computing solution, are interfaces and workflow modules to use Key4hep based simulation and reconstruction programs [145]. The full simulation interface using DDSim [89] already existed in iLCDirac, since it was used extensively for CLIC and ILD studies. New interfaces were integrated for programs to use the parametrised simulation with Delphes, several Monte Carlo generators, such as KKMC, Babayaga, and Bhlumi, and the main Key4hep command `k4run`. The interfaces and workflow modules allow users and the production system to configure the different programs dynamically. For example, iLCDirac must ensure that random seeds for Monte Carlo generators are going to be unique for each job, declare which input files or detector model is used, and register the relevant information as meta data in its file catalogue.

9.2.3 Interface for production managers

While the python interface is convenient for users, the effort to create millions of events with different event types and detectors, and to monitor the success or failure of jobs, can become overwhelming. For this purpose iLCDirac extends the transformation system. With the *dirac-fcc-make-transformation* script, a *production manager* is able to create a simple text file, which instructs the transformation system to create the desired jobs and essentially handle everything from that point forward. Listing 3 shows part of the configuration file to create the same kind of jobs as shown in Listing 2. The configuration file is passed to the *dirac-fcc-make-production* command, which interprets it and creates a *transformation* for each column of *prodIDs* in this example. The transformation system then finds the input files and creates one job for each input file, until all input files have been successfully treated, or reached a maximum number of errors. There is less freedom in the transformation creation, but after defining the desired workflows the only thing that changes for large scale Monte Carlo productions are the parameters present here: the software version, the detector model, and the event types to be simulated. The configuration files to create transformations are additionally stored in a git repository (*https://github.com/HEP-FCC/FCCDIRAC*), which allows reviewing transformations before submission, and sharing of configuration files among the production managers.

9.2.4 Information, traceability

Eventual consistency and reproducibility is mandatory to trust the results of the distributed computing system. When transformations are defined the specific software versions and configuration files for the applications are stored, and cannot be changed once the transformation is submitted. Each job that is created as part of a transformation will then use the exact same software and steering files as all other jobs. The only difference between jobs will be the random seed or if applicable the input file [4]. Based on this knowledge any job that the transformation system created can be run again for debugging purposes. Information about which software and steering

[4] A special case is the background overlay, where randomised background files can be injected into the application's execution.

```
[ddsim]
SteeringFile = cld_steer.py

[Production Parameters]
softwareVersion = key4hep_240412
simulationApplication = ddsim
ProdTypes = Sim
configVersion = key4hep-devel-2
configPackage = fccConfig
detectorModel = CLD_o2_v06

eventsPerJobs = 1000, 1000,  1000,  1000,  1000,  1000,  1000
energies =      240,   240,   240,   240,   240,   240,   240
processes = Htautau,   Hgg,   Hdd,   Huu,   Hss,   Hcc,   Hbb
prodids =     16542, 16545, 16548, 16551, 16554, 16557, 16560
```

Listing 3: Example FCC transformation config

file package is also associated as meta data to all the output files of a transformation. The transformation ID and the specific task ID are also part of the output file name. These two numbers are used to derive the random seed for an application, so even if the application should not store the random seed used, it can be calculated easily given this knowledge. The meta data stored in the file catalogue also contains information about the ancestry and descendants of files. It is possible obtain information about all reconstructed files that resulted from a specific generator sample for example. This meta data information also makes it superfluous to store ancestry information in file-names. Listing 4 shows the meta data and folder hierarchy belonging to a reconstruction transformation. Given a transformation ID, the related detector model or software packages used can be easily obtained. Listing 4 also shows the folder structure that was chosen for the FCC samples. The first folder is simply the virtual organisation (fcc), which is a requirement from the DIRAC system, it is followed by the initial state (ee), the campaign (test_spring2024), centre-of-mass energy (240gev), the event type (mumuH-mH-lower), the detector model (CLD_02_v05), the data type (rec), and finally the transformation ID[5] (00016622).

Using the file catalogue, it is also possible to obtain ancestor and descendent information for files coming from the transformation system. Listing 5 shows an example of the ancestors for a reconstructed file, it shows the simulated file as the parent and the original Monte Carlo record *stdhep* file as the grand-parent.

To make the samples produced by the transformation system easily usable for the analysers, some metadata associated to the files is accumulated by an agent that runs about once per day. It collects the LFN path, the total number of available events, the events per file, the cross-section, its error, generation efficiency, and the responsible production manager. Listing 6 shows the JSON formatted information provided by the agent. This agent directly accesses transformation system for active transformations

[5] For historical reasons this is called *Production* ID or ProdID for short, and the terms transformation and production are used interchangeably

```
- Registered metadata:
    /fcc/ee/test_spring2024/ = {'Campaign': 'test_spring2024'}
    /[snip]/240gev/ = {'Energy': '240'}
    /[snip]/240gev/mumuH-mH-lower/ = {'EvtType':
    'mumuH-mH-lower'}
    /[snip]/240gev/mumuH-mH-lower/CLD_o2_v05 = {'DetectorType':
    'CLD_o2_v05'}
    /[snip]/240gev/mumuH-mH-lower/CLD_o2_v05/rec = {'Datatype':
    'rec'}
    /[snip]/240gev/mumuH-mH-lower/CLD_o2_v05/rec/00016622 =
    {'ProdID': 16622, 'NumberOfEvents': 1000}
- Registered non searchable metadata:
    /[snip]/240gev/mumuH-mH-lower/CLD_o2_v05/rec/00016622 =
    {'SWPackages': 'gaudiapp.key4hep_240412'}
```

Listing 4: Example of meta data registered, and the directory hierarchy of a reconstruction transformation

```
FC:/> ancestor /[snip]/mumuH-mH-lower_rec_16622_91.root -1
/[snip]/mumuH-mH-lower_rec_16622_91.root
1       /[snip]/mumuH-mH-lower_sim_16621_199.root
2            /[snip]/mumuH-mH-lower_stdhep_16620_116.stdhep
```

Listing 5: Ancestor information example.

```
"16622": {
"Status": "Active",
"Version": 0,
"cross-section": 6.77319253587844,
"cross-section-error": 0.0001743518712179634,
"efficiency": 0.04,
"total-number-of-events": 498000,
"number-of-events-per-file": 1000,
"production-manager": "[snip]/CN=Brieuc Francois",
"path":
   "/fcc/[snip]/240gev/mumuH-mH-lower/CLD_o2_v05/rec/00016622"
}
```

Listing 6: Metadata collected for transformation with the ID 16620 in json format.

and file catalogue to find the associated files and their metadata. In the end it uploads a file to the FCC EOS storage, where this file can be read by analysers, or other services without direct access to the iLCDirac system.

10 Outlook

A robust software and computing infrastructure in evaluating the feasibility and potential of the FCC project has been consistently emphasised. To overcome the challenges of resource limitations during the project's early phases, the strategic approach to join a project as Key4hep was adopted, maximizing the use of existing solutions while broadening the applicability of common tools and frameworks to meet the evolving needs of the project. As discussed in Section 1, this approach has resulted in an ecosystem that is now routinely utilised by FCC particle physicists in their daily work. During the next phase consolidation and advancements in all directions will have to occur to ensure suitability for the FCC project needs.

The framework will require continuous consolidation and harmonisation to efficiently support the required workflows. To maximise the use of available computing resources, robust support for multithreading must be guaranteed, and mechanisms to handle heterogeneous resources should be implemented where applicable. These efforts are closely aligned with HL-LHC developments, making it imperative to ensure the seamless integration of those advancements into Key4hep. As Key4hep transitions out of the R&D phase, it is crucial to establish a sustainable support model involving contributions from collaborating institutes. This will ensure the long-term viability and adaptability of the framework to meet evolving project requirements.

The common event data model, EDM4hep, will require ongoing consolidation and refinement, particularly in response to the evolving needs of the developing detector concepts. In this context, an ongoing pilot project for migrating LEP data to EDM4hep could prove invaluable. This initiative not only tests EDM4hep with real, non-simulated data but also offers FCC physicists a unique opportunity to gain hands-on experience by analysing actual experimental data. However, to fully realise its potential and make it broadly useful for the community, this initial effort will require additional resources and further development.

On the event generator side, not covered in this note, further efforts are required to enhance the accuracy of simulations for beam-related quantities, such as beam energy spread, crossing-angle spread, bunch length, transverse size, and final-state particle deflection caused by colliding bunches. Additionally, improvements in modelling initial-state and final-state radiation, including their interference, are essential to address potential challenges in achieving the statistical precision anticipated at FCC-ee, particularly at the Z pole. To ensure the applicability of these advancements across all Monte Carlo generator codes relevant to FCC-ee, establishing close and sustained collaboration with the code authors will be critical. This collaborative approach will help integrate these technologies effectively and align them with the high-precision requirements of the project.

Significant progress have been made in the development of full simulation tools during the Feasibility Study; however, much remains to be accomplished during the pre-TDR phase. A critical task will be implementing detailed digitisers that incorporate all the effects enabling a realistic assessment of beam-induced backgrounds' impact on detector performance. This impact will then need to be propagated to key physics studies. To consolidate the achievements of the results produced during the Feasibility Study, the complete SIM-DIGI-RECO chain must be established

for all detector concepts currently under consideration. Additionally, implementing advanced reconstruction techniques such as particle-flow will be crucial for providing the necessary tools to optimise these detector concepts. Advanced AI approaches will play a pivotal role in this effort, offering superior performance compared to heuristic algorithms and facilitating rapid re-optimisation in response to changes, such as modifications to detector geometry.

As new sub-detector technologies will emerge, their geometry modelling will need to be integrated into existing detector concepts. Entirely new detector configurations, distinct from those studied during the Feasibility Study, will also be explored. Achieving this will require enhancing the flexibility of geometry implementations and improving sub-detector interoperability by generalising reconstruction algorithms. If successful, this effort will empower the community to identify optimal detector designs for the project with minimal human resource expenditure.

To determine the most suitable detector scenarios, metrics should be based on results from physics studies rather than isolated detector performance measures. This approach will ensure a holistic consideration of sub-detector interplay. In this context, improving the interface between centrally produced samples and analysis frameworks, including Python-based frameworks, is essential. Specific areas for investigation include enhancing the computing performance of the PODIO RDataSource and implementing mechanisms to facilitate the production of flat ROOT tuples. The content of these tuples as well as their production workflow will have to be designed to meet the needs of most analyses.

To deliver all the relevant full simulation-based studies by the end of the pre-TDR phase, the FCC PED studies will demand a significant expansion of both computing and human resources to achieve their objectives effectively. Meeting FCC's growing computational needs involves pursuing multiple avenues inspired by the LHC model, with integration into the WLCG resource pool, and leveraging HPC calls and exploring opportunistic usage of available resources. Should the acquisition of additional resources prove challenging, mitigation techniques have been identified and will need to be implemented.

The substantial progress achieved in the software ecosystem during the FCC Feasibility Study has demonstrated that a dedicated software core team at CERN, integrated with fruitful collaborative work with the other work packages of the PED activities, are pivotal in driving and coordinating the majority of FCC software activities. Consolidating and expanding the team is essential to sustain progress, address emerging challenges, and capitalise on synergies with other initiatives. Closer ties with the LHC community can unlock mutual benefits, by leveraging shared technologies that advance both FCC and LHC objectives. Continued efforts are needed to attract more external contributors from institutes worldwide, raise awareness and interest in FCC among the broader scientific community, encouraging partnerships and resource sharing.

References

[1] M. Frank, F. Gaede, M. Petric, A. Sailer. AIDASoft/DD4hep (2018). https://doi.org/10.5281/zenodo.592244. Webpage: http://dd4hep.cern.ch/

[2] The CEPC Study Group. CEPC Conceptual Design Report: Volume 1 - Accelerator (2018). URL https://arxiv.org/abs/1809.00285

[3] P. Burrows, et al., The Compact Linear Collider (CLIC) - 2018 Summary Report p. 112 (2018). https://doi.org/10.23731/CYRM-2018-002. arXiv:1812.06018 [physics.acc-ph]

[4] P. Bambade, et al., The International Linear Collider: A Global Project p. 104 (2019). https://doi.org/10.48550/arXiv.1903.01629. arXiv:1903.01629 [physics.hep-ex]

[5] Q. Luo, W. Gao, J. Lan, W. Li, D. Xu, Progress of Conceptual Study for the Accelerators of a 2-7GeV Super Tau Charm Facility at China (10), 643–645 (2019). https://doi.org/doi:10.18429/JACoW-IPAC2019-MOPRB031. URL http://jacow.org/ipac2019/papers/moprb031.pdf. Https://doi.org/10.18429/JACoW-IPAC2019-MOPRB031

[6] G. Ganis, C. Helsens, V. Völkl, Key4hep, a framework for future HEP experiments and its use in FCC. Eur. Phys. J. Plus **137**(1), 149 (2022). https://doi.org/10.1140/epjp/s13360-021-02213-1. arXiv:2111.09874 [hep-ex]

[7] Future HEP projects community. Future Collider Software Workshop (2019). URL https://agenda.infn.it/event/19047. Held in Bologna, Italy

[8] Future HEP projects community. Mini-workshop: Experiment/Detector - Software and Physics Requirements for e+e- Colliders (2020). URL http://iasprogram.ust.hk/hep/2020/. Held in Hong Kong

[9] M. Clemencic, et al., Gaudi evolution for future challenges (2017). https://doi.org/doi:10.1088/1742-6596/898/4/042044. J. Phys. Conf. Ser. 898 (2017) no.4, 042044

[10] R. Brun, F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta, V. Vassilev, S. Linev, D. Piparo, G. Ganis, B. Bellenot, E. Guiraud, G. Amadio, wverkerke, P. Mato, TimurP, M. Tadel, wlav, E. Tejedor, J. Blomer, A. Gheata, S. Hageboeck, S. Roiser, marsupial, S. Wunsch, O. Shadura, A. Bose, CristinaCristescu, X. Valls, R. Isemann. root-project/root: v6.18/02 (2020). https://doi.org/10.5281/zenodo.3895860. URL https://doi.org/10.5281/zenodo.3895860

[11] J. Allison, et al. Geant4 developments and applications. https://doi.org/doi:10.1109/TNS.2006.869826. IEEE Trans. Nucl. Sci. 53 (2006), 270

[12] F. Gaede, et al. PODIO: recent developments in the Plain Old Data EDM toolkit. https://doi.org/10.1051/epjconf/202024505024. EPJ Web Conf. 245 (2020), 05024

[13] F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer, G.A. Stewart, V. Volkl, J. Wang, EDM4hep and podio - The event data model of the Key4hep project and its implementation. EPJ Web Conf. **251**, 03026 (2021). https://doi.org/10.1051/epjconf/202125103026

[14] F. Gaede, T. Madlener, P. Declara Fernandez, G. Ganis, B. Hegner, C. Helsens, A. Sailer, G.A. Stewart, V. Voelkl, EDM4hep - a common event data model for HEP experiments. PoS **ICHEP2022**, 1237 (2022). https://doi.org/10.22323/1.414.1237

[15] F. Gaede, T. Behnke, N. Graf, T. Johnson, LCIO: A Persistency framework for linear collider simulation studies. eConf **C0303241**, TUKT001 (2003). arXiv:physics/0306114

[16] J.M. Carceller, F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer, G.A. Stewart, V. Volkl, Towards podio v1.0 - A first stable release of the EDM toolkit. EPJ Web Conf. **295**, 06018 (2024). https://doi.org/10.1051/epjconf/202429506018. arXiv:2312.08206 [hep-ex]

[17] F. Gaede, B. Hegner, P. Mato, PODIO: An Event-Data-Model Toolkit for High Energy Physics Experiments. J. Phys. Conf. Ser. **898**(7), 072039 (2017). https://doi.org/10.1088/1742-6596/898/7/072039

[18] J. Blomer, P. Canal, A. Naumann, D. Piparo, Evolution of the ROOT Tree I/O. EPJ Web Conf. **245**, 02030 (2020). https://doi.org/10.1051/epjconf/202024502030. arXiv:2003.07669 [cs.DB]

[19] Various authors. key4hep/k4fwcore. https://github.com/key4hep/k4fwcore. [Online]

[20] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. de Supinski, S. Futral, (Austin, Texas, US, 2015), Supercomputing 2015 (SC'15). URL https://tgamblin.github.io/pubs/spack-sc15.pdf

[21] Various authors. cvmfs/cvmfs. https://github.com/cvmfs/cvmfs. [Online]

[22] Various authors. key4hep/key4hep-spack. https://github.com/key4hep/key4hep-spack. [Online]

[23] Various authors. key4hep/key4hep-actions. https://github.com/key4hep/key4hep-actions. [Online]

[24] M. Frank, F. Gaede, C. Grefe, P. Mato, DD4hep: A Detector Description Toolkit for High Energy Physics Experiments. J. Phys. Conf. Ser. **513**(2), 022010 (2014). https://doi.org/10.1088/1742-6596/513/2/022010

[25] M. Petric, M. Frank, F. Gaede, A. Sailer, New developments in DD4hep. EPJ Web Conf. **214**, 02037 (2019). https://doi.org/10.1051/epjconf/201921402037

[26] F. Gaede, M. Frank, M. Petric, A. Sailer, DD4hep a community driven detector description for HEP. EPJ Web Conf. **245**, 02004 (2020). https://doi.org/10.1051/epjconf/202024502004. URL https://cds.cern.ch/record/2719129

[27] M. Frank, F. Gaede, N. Nikiforou, M. Petric, A. Sailer, Ddg4: A simulation framework using the dd4hep detector description toolkit. J. Phys. Conf. Ser. **664**, 072017 (2015). https://doi.org/10.1088/1742-6596/664/7/072017

[28] M. Frank, F. Gaede, M. Petric, A. Sailer, Conditions and alignment extensions of the DD4hep detector description toolkit. EPJ Web Conf. **214**, 02042 (2019). https://doi.org/10.1051/epjconf/201921402042

[29] A. Sailer, M. Frank, F. Gaede, D. Hynds, S. Lu, N. Nikiforou, M. Petric, R. Simoniello, G.G. Voutsinas, DD4Hep based event reconstruction. J. Phys. Conf. Ser. **898**(4), 042017 (2017). https://doi.org/10.1088/1742-6596/898/4/042017

[30] J. Allison, et al., Recent developments in Geant4. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **835**, 186–225 (2016). https://doi.org/https://doi.org/10.1016/j.nima.2016.06.125. URL https://www.sciencedirect.com/science/article/pii/S0168900216306957

[31] S. Agostinelli, et al., GEANT4: A Simulation toolkit. Nucl. Instrum. Meth. **A506**, 250–303 (2003). https://doi.org/10.1016/S0168-9002(03)01368-8

[32] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, M. Selvaggi, DELPHES 3, A modular framework for fast simulation of a generic collider experiment. JHEP **02**, 057 (2014). https://doi.org/10.1007/JHEP02(2014)057. arXiv:1307.6346 [hep-ex]

[33] `TrackCovariance` module in Delphes. https://github.com/delphes/delphes/blob/master/modules/TrackCovariance.cc

[34] `TimeOfFlight` module in Delphes. https://github.com/delphes/delphes/blob/master/modules/TimeOfFlight.cc

[35] `ClusterCounting` module in Delphes. https://github.com/delphes/delphes/blob/master/modules/ClusterCounting.cc

[36] Delphes release. https://github.com/delphes/delphes

[37] FCC-Config IDEA detector Delphes card. https://github.com/HEP-FCC/FCC-config/blob/winter2023/FCCee/Delphes/card_IDEA.tcl

[38] FCC-Config IDEA with Silicon CLD-like tracking card. https://github.com/HEP-FCC/FCC-config/blob/winter2023/FCCee/Delphes/card_IDEA_SiTracking.tcl

[39] S. Aumiller, D. Garcia, M. Selvaggi. Jet Flavor Tagging Performance at FCC-ee (2024). https://doi.org/10.17181/8g834-jv464

[40] F. Bedeschi, L. Gouskos, M. Selvaggi, Jet flavour tagging for future colliders with fast simulation. Eur. Phys. J. C **82**(7), 646 (2022). https://doi.org/10.1140/epjc/s10052-022-10609-1. arXiv:2202.03285 [hep-ex]

[41] A. Abada, et al., FCC-ee: The Lepton Collider: Future Circular Collider Conceptual Design Report Volume 2. Eur. Phys. J. ST **228**(2), 261–623 (2019). https://doi.org/10.1140/epjst/e2019-900045-4

[42] M. Aleksa, F. Bedeschi, R. Ferrari, F. Sefkow, C.G. Tully, Calorimetry at FCC-ee. Eur. Phys. J. Plus **136**(10), 1066 (2021). https://doi.org/10.1140/epjp/s13360-021-02034-2. arXiv:2109.00391 [hep-ex]

[43] R. Veenhof, GARFIELD, recent developments. Nucl. Instrum. Meth. A **419**, 726–730 (1998). https://doi.org/10.1016/S0168-9002(98)00851-1

[44] M. Cacciari, G.P. Salam, G. Soyez, FastJet User Manual. Eur. Phys. J. C **72**, 1896 (2012). https://doi.org/10.1140/epjc/s10052-012-1896-2. arXiv:1111.6097 [hep-ph]

[45] S. Catani, Y.L. Dokshitzer, M. Olsson, G. Turnock, B.R. Webber, New clustering algorithm for multi-jet cross-sections in e+ e- annihilation. Phys. Lett. B **269**, 432–438 (1991). https://doi.org/10.1016/0370-2693(91)90196-W

[46] T. Madlener, V. Volkl, J.M. Carceller, C. Helsens, M. Chrzaszcz, F. Gaede, B. Stapf, M. Selvaggi, EmanuelPerez. key4hep/k4simdelphes (2024). https://doi.org/10.5281/zenodo.4564682. URL https://doi.org/10.5281/zenodo.4564682

[47] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, An Introduction to PYTHIA 8.2. Comput. Phys. Commun. **191**, 159–177 (2015). https://doi.org/10.1016/j.cpc.2015.01.024. arXiv:1410.3012 [hep-ph]

[48] D.J. Lange, The EvtGen particle decay simulation package. Nucl. Instrum. Meth. A **462**, 152–155 (2001). https://doi.org/10.1016/S0168-9002(01)00089-4

[49] A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski, A. Verbytskyi, The HepMC3 event record library for Monte Carlo event generators. Comput. Phys. Commun. **260**, 107310 (2021). https://doi.org/10.1016/j.cpc.2020.107310. arXiv:1912.08005 [hep-ph]

[50] Various authors. key4hep/k4geo. https://github.com/key4hep/k4geo. [Online]

[51] M. Boscolo, F. Palla, F. Bosi, F. Fransesini, S. Lauciani, Mechanical model for the FCC-ee interaction region. EPJ Techniques and Instrumentation **10**(1), 16 (2023). https://doi.org/10.1140/epjti/s40485-023-00103-7. URL https://doi.org/10.1140/epjti/s40485-023-00103-7

[52] A. Novokhatski, et al., Estimated heat load and proposed cooling system in the FCC-ee Interaction region beam pipe (14), 260–263 (2023). https://doi.org/10.18429/JACoW-IPAC2023-MOPA092. URL https://indico.jacow.org/event/41/contributions/1316

[53] N. Alipour Tehrani, J.J. Blaising, B. Cure, D. Dannheim, F. Duarte Ramos, K. Elsener, A. Gaddi, H. Gerwig, S. Green, C. Grefe, D. Hynds, W. Klempt, L. Linssen, N. Nikiforou, A.M. Nurnberg, J.S. Marshall, M. Petric, S. Redford, P.G. Roloff, A. Sailer, F. Sefkow, E. Sicking, N. Siegrist, F.R. Simon, R. Simoniello, S. Spannagel, S.K. Sroka, L.R. Strom, M.A. Weber, CLICdet: The post-CDR CLIC detector model (2017). URL https://cds.cern.ch/record/2254048

[54] N. Bacchetta, J.J. Blaising, E. Brondolin, M. Dam, D. Dannheim, K. Elsener, D. Hynds, P. Janot, A.M. Kolano, E. Leogrande, L. Linssen, A.M. Nurnberg, E.F. Perez, M. Petric, P.G. Roloff, A. Sailer, N. Siegrist, O. Viazlo, G.G. Voutsinas, M.A. Weber, CLD - A Detector Concept for the FCC-ee. Tech. rep., CERN, Geneva (2019). URL http://cds.cern.ch/record/2697140. 75 pages, 67 figures

[55] N. Alipour Tehrani, Simulation and tracking studies for a drift chamber at the FCC-ee experiment. Tech. rep., CERN, Geneva (2019). URL https://cds.cern.ch/record/2670936

[56] W. Elmetenawee, g. chiarello, A. Corvaglia, F. Cuna, N. De Filippis, E. Gorini, F. Grancagnolo, M. Maggi, A. Miccoli, M. Panareo, M. Primavera, G. Francesco Tassielli, A. Ventura, in *Proceedings of 41st International Conference on High Energy physics — PoS(ICHEP2022)* (Sissa Medialab, 2022), ICHEP2022, p. 362. https://doi.org/10.22323/1.414.0362. URL http://dx.doi.org/10.22323/1.414.0362

[57] A. Ilg, in *Proceedings of The European Physical Society Conference on High Energy Physics — PoS(EPS-HEP2023)* (Sissa Medialab, 2023), EPS-HEP2023, p. 600. https://doi.org/10.22323/1.449.0600. URL http://dx.doi.org/10.22323/1.449.0600

[58] G. Voutsinas, K. Elsener, P. Janot, D. El Khechen, A. Kolano, E. Leogrande, E.F. Perez, N.A. Tehrani, O. Viazlo, M. Boscolo, O. Blanco, F. Collamati, N. Bacchetta, M. Dam, M.K. Sullivan, Fcc-ee interaction region backgrounds. International Journal of Modern Physics A **35**(15n16), 2041009 (2020). https://doi.org/10.1142/s0217751x20410092. URL http://dx.doi.org/10.1142/S0217751X20410092

[59] Boscolo, Manuela, Frasca, Alessandro, Novokhatski, Alexander, Ciarma, Andrea, Lechner, Anton, Ilg, Armin, Di Pasquale, Enrico, Palla, Fabrizio, Bosi, Filippo, Fransesini, Francesco, Zimmermann, Frank, Broggi, Giacomo, Nigrelli, Giulia, Lerner, Giuseppe, Burkhardt, Helmut, Seeman, John, Oide, Katsunobu, André, Kévin, Benedikt, Michael, Koratzinos, Michael, Raimondi, Pantaleo, Bruce, Roderik, Lauciani, Stefano, Raubenheimer, Tor, Progress in the design of the future circular collider FCC-ee interaction region (2024). https://doi.org/10.18429/JACOW-IPAC2024-TUPC67. URL https://jacow.org/ipac2024/doi/jacow-ipac2024-tupc67

[60] L. Pancheri, R.A. Giampaolo, A.D. Salvo, S. Mattiazzo, T. Corradino, P. Giubilato, R. Santoro, M. Caccia, G. Margutti, J.E. Olave, M. Rolo, A. Rivetti, Fully Depleted MAPS in 110-nm CMOS Process With 100–300-μm Active Substrate. IEEE Transactions on Electron Devices **67**(6), 2393–2399 (2020). https://doi.org/10.1109/ted.2020.2985639. URL http://dx.doi.org/10.1109/TED.2020.2985639

[61] I. Peric, A. Andreazza, H. Augustin, M. Barbero, M. Benoit, R. Casanova, F. Ehrler, G. Iacobucci, R. Leys, A.M. Gonzalez, P. Pangaud, M. Prathapan, R. Schimassek, A. Schoning, E.V. Figueras, A. Weber, M. Weber, W. Wong, H. Zhang, High-Voltage CMOS Active Pixel Sensor. IEEE Journal of Solid-State Circuits **56**(8), 2488–2502 (2021). https://doi.org/10.1109/jssc.2021.3061760. URL http://dx.doi.org/10.1109/JSSC.2021.3061760

[62] M. Frank, F. Gaede, M. Petrič, A. Sailer, CAD support and new developments in DD4hep. EPJ Web of Conferences **251**, 03015 (2021). https://doi.org/10.1051/epjconf/202125103015. URL http://dx.doi.org/10.1051/epjconf/202125103015

[63] L. Freitag. Benefits of Minimizing the Vertex Detector Material Budget at the FCC-ee (2023). URL https://cds.cern.ch/record/2851362. Presented 01 Feb 2023

[64] The ALICE collaboration, Technical Design report for the ALICE Inner Tracking System 3 - ITS3 ; A bent wafer-scale monolithic pixel detector. Tech. rep., CERN, Geneva (2024). URL https://cds.cern.ch/record/2890181. Co-project Manager: Magnus Mager, magnus.mager@cern.chds

[65] The ALICE Collaboration. Letter of intent for ALICE 3: A next-generation heavy-ion experiment at the LHC (2022). URL https://arxiv.org/abs/2211.02491

[66] G.F. Tassielli, A proposal of a drift chamber for the IDEA experiment for a future e+e- collider. PoS **ICHEP2020**, 877 (2021). https://doi.org/10.22323/1.390.0877

[67] L. Cerrito, M. Gatta, L. Paoluzi, E. Santovetti, Particle identification using cluster counting in a large drift chamber at normal conditions. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **434**(2), 261–270 (1999). https://doi.org/https://doi.org/10.1016/S0168-9002(99)00551-3. URL https://www.sciencedirect.com/science/article/pii/S0168900299005513

[68] A. Tolosa-Delgado, B. Francois. Detector Driver for DriftChamber_o1_v02. https://github.com/key4hep/k4geo/blob/main/detector/tracker/DriftChamber_o1_v02.cpp (2024). [Online; accessed 8-July-2024]

[69] K. Hoshina, K. Fujii, O. Nitoh, Development of a Geant4 solid for stereo mini-jet cells in a cylindrical drift chamber. Computer Physics Communications **153**(3), 373–391 (2003). https://doi.org/https://doi.org/10.1016/S0010-4655(03)00206-6. URL https://www.sciencedirect.com/science/article/pii/S0010465503002066

[70] A. Tolosa-Delgado, B. Francois. Data extension for DriftChamber_o1_v02. https://github.com/AIDASoft/DD4hep/blob/master/DDRec/include/DDRec/DCH_info.h (2024). [Online; accessed 8-July-2024]

[71] W. Elmetenawee, et al., The Tracking performance for the IDEA drift chamber. PoS **ICHEP2022**, 362 (2022). https://doi.org/10.22323/1.414.0362. arXiv:2211.12568 [physics.ins-det]

[72] R. Forty, RICH pattern recognition for LHCb. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **433**(1), 257–261 (1999). https://doi.org/https://doi.org/10.1016/S0168-9002(99)00310-1. URL https://www.sciencedirect.com/science/article/pii/S0168900299003101

[73] M. Battaglia, P. M. Kluit, Particle identification using the DELPHI RICH detectors. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **433**(1), 252–256 (1999). https://doi.org/https://doi.org/10.1016/S0168-9002(99)00464-7. URL https://www.sciencedirect.com/science/article/pii/S0168900299004647

[74] M.S. Stojke. Differential Evolution algorithm for general purpose optimization. https://github.com/milsto/differential-evolution/tree/master (2023). [Online; accessed 8-July-2024]

[75] A. Tolosa-Delgado, M. Tat, R. Forty. Detector Driver for ARC_geo_o1_v01. https://github.com/key4hep/k4geo/blob/main/detector/PID/ARC_geo_o1_v01.cpp

(2023). [Online; accessed 8-July-2024]

[76] J. Aguilar, B. Pawlik, S.Kulis, M. Idzik, M. Chrzaszcz, W. Daniluk, E. Kielar, J. Kotula, A. Moszczynski, K. Oliwa, W. Wierba, L. Zawiejski, K. Afanaciev, H. Henschel, A. Ignatenko, S. Kollowa, W. Lohmann, O. Novgorodova, S. Schuwalow, I. Levy, Luminometer for the future International Linear Collider -Simulation and Beam Test Results. Physics Procedia **37**, 258–265 (2012). https://doi.org/https://doi.org/10.1016/j.phpro.2012.02.373. URL https://www.sciencedirect.com/science/article/pii/S1875389212016884. Proceedings of the 2nd International Conference on Technology and Instrumentation in Particle Physics (TIPP 2011)

[77] G. Marchiori. giovannimarchiori/calodisplay. https://github.com/giovannimarchiori/calodisplay. [Online]

[78] Wonyong Chung, Differentiable Full Detector Simulation of a Projective Dual-Readout Crystal Electromagnetic Calorimeter with Longitudinal Segmentation and Precision Timing. EPJ Web Conf. **320**, 00052 (2025). https://doi.org/10.1051/epjconf/202532000052. URL https://doi.org/10.1051/epjconf/202532000052

[79] Wonyong Chung. wonyongc/SCEPCal. https://github.com/wonyongc/SCEPCal. Repository for SCEPCal simulation

[80] S. Lee, M. Livan, R. Wigmans, Dual-readout calorimetry. Rev. Mod. Phys. **90**, 025002 (2018). https://doi.org/10.1103/RevModPhys.90.025002. URL https://link.aps.org/doi/10.1103/RevModPhys.90.025002

[81] N. Sultanova, S. Kasarova, I. Nikolov, Dispersion Properties of Optical Polymers. ACTA PHYSICA POLONICA A **116**, 585–587 (2009). https://doi.org/10.12693/APhysPolA.116.585

[82] T. Kaino, M. Fujiki, S. Nara, Low-loss polystyrene core-optical fibers. Journal of Applied Physics **52**(12), 7061–7063 (1981). https://doi.org/10.1063/1.328702. URL https://doi.org/10.1063/1.328702

[83] C. Alfieri, A.B. Rodrigues Cavalcante, C. Joram, A set-up to measure the optical attenuation length of scintillating fibres. Tech. rep., CERN, Geneva (2015). URL https://cds.cern.ch/record/2011565

[84] G. Beadie, M. Brindza, R.A. Flynn, A. Rosenberg, J.S. Shirk, Refractive index measurements of poly(methyl methacrylate) (pmma) from $0.4 - 1.6\mu m$. Appl. Opt. **54**(31), F139–F143 (2015). https://doi.org/10.1364/AO.54.00F139. URL http://opg.optica.org/ao/abstract.cfm?URI=ao-54-31-F139

[85] S. Abrate, in *Handbook of Fiber Optic Data Communication (Fourth Edition)*, ed. by C. DeCusatis, fourth edition edn. (Academic Press, Oxford, 2013), pp. 37–54. https://doi.org/https://doi.org/10.1016/B978-0-12-401673-6.00003-9. URL https://www.sciencedirect.com/science/article/pii/B9780124016736000039

[86] A.C.J. Peed, *KODAK Photographic Filters Handbook* (Eastman Kodak Company, 1992), p. 127

[87] Various authors. HEP-FCC/FCC-config. https://github.com/HEP-FCC/FCC-config/tree/main/FCCee/FullSim. [Online]

[88] Various authors. key4hep/CLDConfig. https://github.com/key4hep/CLDConfig/. [Online]

[89] M. Petric, M. Frank, F. Gaede, S. Lu, N. Nikiforou, A. Sailer, Detector simulations with DD4hep. J. Phys. Conf. Ser. **898**(4), 042015 (2016). https://doi.org/10.1088/1742-6596/898/4/042015

[90] N. Ampilogov, S. Cometti, J. Agarwala, V. Chmill, R. Ferrari, G. Gaudio, P. Giacomelli, A. Giaz, A. Karadzhinova-Ferrer, A. Loeschcke-Centeno, A. Negri, L. Pezzotti, G. Polesello, E. Proserpio, A. Ribon, R. Santoro, I. Vivarelli, Exposing a fibre-based dual-readout calorimeter to a positron beam. Journal of Instrumentation (2023). https://doi.org/https://doi.org/10.1088/1748-0221/18/09/P09021. URL https://iopscience.iop.org/article/10.1088/1748-0221/18/09/P09021

[91] F. Cuna, N. De Filippis, F. Grancagnolo, G.F. Tassielli. Simulation of particle identification with the cluster counting technique (2021). URL https://arxiv.org/abs/2105.07064

[92] Various authors. ILCSoft/DDMarlinPandora. https://github.com/iLCSoft/DDMarlinPandora. [Online]

[93] Various authors. HEP-FCC/k4ReCalorimeter. https://github.com/HEP-FCC/k4RecCalorimeter. [Online]

[94] F. Simon, Silicon photomultipliers in particle and nuclear physics. Nucl. Instrum. Meth. A **926**, 85–100 (2019). https://doi.org/https://doi.org/10.1016/j.nima.2018.11.042. URL https://www.sciencedirect.com/science/article/pii/S0168900218316176. Silicon Photomultipliers: Technology, Characterisation and Applications

[95] E. Proserpio, R. Santoro, *SimSiPM: a library for SiPM simulation.* Como, Italy (2021). URL https://github.com/EdoPro98/SimSiPM

[96] F. Didierjean, G. Duchêne, A. Lopez-Martens, The Deterministic Annealing Filter: A new clustering method for γ-ray tracking algorithms. Nuclear Instruments

and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **615**(2), 188–200 (2010). https://doi.org/https://doi.org/10.1016/j.nima.2010.01.030. URL https://www.sciencedirect.com/science/article/pii/S0168900210000987

[97] E. Brondolin, E. Leogrande, D. Hynds, F. Gaede, M. Petrič, A. Sailer, R. Simoniello, Conformal tracking for all-silicon trackers at future electron–positron colliders. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **956**, 163304 (2020). https://doi.org/10.1016/j.nima.2019.163304. URL http://dx.doi.org/10.1016/j.nima.2019.163304

[98] Dolores Garcia, Michele Selvaggi, Brieuc Francois. Geometric Graph Neural Network based track finding. https://repository.cern/records/pwrx1-wvn43. [Online]

[99] Marchiori, Giovanni. gmarchio/fcc-lar-energy-calibration. https://gitlab.cern.ch/gmarchio/fcc-lar-energy-calibration. [Online]

[100] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems **30**, 3146–3154 (2017)

[101] Marchiori, Giovanni. gmarchio/fcc-lar-photonid. https://gitlab.cern.ch/gmarchio/fcc-lar-photonid. [Online]

[102] M. Thomson, Particle flow calorimetry and the PandoraPFA algorithm. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **611**(1), 25–40 (2009). https://doi.org/10.1016/j.nima.2009.09.009. URL http://dx.doi.org/10.1016/j.nima.2009.09.009

[103] R. Pöschl, The CALICE SiW ECAL Technological Prototype—Status and Outlook. Instruments **6**(4), 75 (2022). https://doi.org/10.3390/instruments6040075. URL http://dx.doi.org/10.3390/instruments6040075

[104] M. Aleksa, F. Bedeschi, R. Ferrari, F. Sefkow, C.G. Tully. Calorimetry at FCC-ee (2021). URL https://arxiv.org/abs/2109.00391

[105] Various authors. Key4hep Nobel Liquid ECal source file. https://github.com/key4hep/k4geo/blob/559f41185a8f166519a392ff7cea3ef0add41b0b/detector/calorimeter/ECalBarrel_NobleLiquid_InclinedTrapezoids_o1_v01_geo.cpp#L564C1-L632C1. [Online]

[106] PandoraPFA on allegro. URL https://indico.cern.ch/event/1481286/contributions/6243894/attachments/2975415/5237614/FCC_FullSim_ALLEGRO_PandoraPFA_slides.pdf

[107] R.J.G.B. Campello, D. Moulavi, J. Sander, Density-based clustering based on hierarchical density estimates pp. 160–172 (2013)

[108] Dolores Garcia, Gregor Krzmanc, Michele Selvaggi. Machine Learning based Particle Flow. https://repository.cern/records/n9wc2-09n03. [Online]

[109] C. Helsens, E. Perez, M. Selvaggi, V. Volkl, L. Forthomme, J. Munch Torndal. Hep-fcc/fccanalyses: v0.9.0 (2024). https://doi.org/10.5281/zenodo.10693709. URL https://doi.org/10.5281/zenodo.10693709

[110] E. Guiraud, A. Naumann, D. Piparo. TDataFrame: functional chains for ROOT data analyses (2017). https://doi.org/10.5281/zenodo.260230. URL https://doi.org/10.5281/zenodo.260230

[111] E. Perez, C. Helsens, P. Azzi, D. Hill, R. Gonzalez Suarez, V. Volkl. Hep-fcc/fcceephysicsperformance: spring2021_bc2taunu_patch01 (2021). https://doi.org/10.5281/zenodo.4795422. URL https://doi.org/10.5281/zenodo.4795422

[112] Hep-fcc/fcchhphysicsperformance (2025). URL https://github.com/HEP-FCC/FCChhPhysicsPerformance

[113] RDataSource (2025). URL https://root.cern.ch/doc/master/classROOT_1_1RDF_1_1RDataSource.html

[114] J. Eysermans, G. Bernardi, L. Ang. Higgs boson mass and model-independent ZH cross-section at FCC-ee in the di-electron and di-muon final states (2024). https://doi.org/10.17181/a68b8-3mt57. URL https://doi.org/10.17181/a68b8-3mt57

[115] A. Del Vecchio, L. Gouskos, G. Marchiori, M. Selvaggi. Measurement of Higgs boson hadronic decays with Z(vv/ll)H events at FCC-ee at $\sqrt{s} = 240$ GeV (2023). https://doi.org/10.17181/rxdmc-4sa60. URL https://doi.org/10.17181/rxdmc-4sa60

[116] H. Qu, L. Gouskos, ParticleNet: Jet Tagging via Particle Clouds. Phys. Rev. D **101**(5), 056019 (2020). https://doi.org/10.1103/PhysRevD.101.056019. arXiv:1902.08570 [hep-ph]

[117] T. Chen, C. Guestrin, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2016), KDD '16, p. 785–794. https://doi.org/10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785

[118] A. Hayrapetyan, et al., The CMS statistical analysis and combination tool: Combine (2024). Submitted to *Comput. Softw. Big Sci.* arXiv:2404.06614 [physics.data-an]

[119] J. Bezanson, A. Edelman, S. Karpinski, V.B. Shah, Julia: A fresh approach to numerical computing. SIAM review **59**(1), 65–98 (2017). URL https://doi.org/10.1137/141000671

[120] L. Gray, N. Smith, A. Novak, P. Fackeldey, B. Tovar, Y.M. Chen, G. Watts, I. Krommydas. coffea (2024). https://doi.org/10.5281/zenodo.13942127. URL https://doi.org/10.5281/zenodo.13942127

[121] FCC Software (2024). URL https://cern.ch/fccsw

[122] JSROOT (2024). URL https://root.cern.ch/js/

[123] iLCSoft/CED (2024). URL https://github.com/iLCSoft/CED

[124] M. Tadel, EVE: Event visualization environment of the ROOT framework. PoS **ACAT08**, 103 (2008). https://doi.org/10.22323/1.070.0103

[125] Geometry Description Markup Language (GDML) (2024). URL https://gdml.web.cern.ch/GDML/

[126] "Phoenix", an experiment independent web-based event display for High Energy Physics (2024). URL https://hepsoftwarefoundation.org/phoenix/

[127] glTF: The 3D Asset Delivery Format (2024). URL https://www.khronos.org/Gltf

[128] Eede: EDM4hep Event Data Explorer (2024). URL https://github.com/key4hep/eede

[129] Pixi.js: The HTML5 Creation Engine (2024). URL https://pixijs.com/

[130] Scene Graph (2024). URL https://pixijs.com/8.x/guides/basics/scene-graph#culling. See subsection: Culling

[131] WebGPU (2024). URL https://www.w3.org/TR/webgpu/

[132] WebAssembly (2024). URL https://webassembly.org/

[133] C. Helsens, E. Perez, M. Selvaggi. HEP-FCC/EventProducer: spring2021_Bc2TauNu (2021). https://doi.org/10.5281/zenodo.4817856. URL https://doi.org/10.5281/zenodo.4817856

[134] S. Jadach, B.F.L. Ward, Z. Wąs, The Precision Monte Carlo event generator K K for two fermion final states in e+ e- collisions. Comput. Phys. Commun. **130**, 260–325 (2000). https://doi.org/10.1016/S0010-4655(00)00048-5. arXiv:hep-ph/9912214

[135] S. Jadach, B.F.L. Ward, Z. Wąs, S.A. Yost, A. Siodmok, Multi-photon Monte Carlo event generator KKMCee for lepton and quark pair production in lepton colliders. Comput. Phys. Commun. **283**, 108556 (2023). https://doi.org/10.1016/j.cpc.2022.108556. arXiv:2204.11949 [hep-ph]

[136] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.S. Shao, T. Stelzer, P. Torrielli, M. Zaro, The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. JHEP **07**, 079 (2014). https://doi.org/10.1007/JHEP07(2014)079. arXiv:1405.0301 [hep-ph]

[137] W. Kilian, T. Ohl, J. Reuter, WHIZARD: Simulating Multi-Particle Processes at LHC and ILC. Eur. Phys. J. C **71**, 1742 (2011). https://doi.org/10.1140/epjc/s10052-011-1742-y. arXiv:0708.4233 [hep-ph]

[138] M. Moretti, T. Ohl, J. Reuter, O'Mega: An Optimizing matrix element generator pp. 1981–2009 (2001). arXiv:hep-ph/0102195

[139] T. Sjostrand, S. Mrenna, P.Z. Skands, PYTHIA 6.4 Physics and Manual. JHEP **05**, 026 (2006). https://doi.org/10.1088/1126-6708/2006/05/026. arXiv:hep-ph/0603175

[140] B. Francois, C. Helsens, G. Marchiori, G. GANIS, T. Li, J.M. Carceller. Hep-fcc/fcc-config: v0.1.0 (2024). https://doi.org/10.5281/zenodo.4783069. URL https://doi.org/10.5281/zenodo.4783069

[141] URL https://dirac.readthedocs.io

[142] A. Tsaregorodtsev, M. Bargiotti, N. Brook, A.C. Ramo, G. Castellani, P. Charpentier, C. Cioffi, J. Closier, R.G. Díaz, G. Kuznetsov, Y.Y. Li, R. Nandakumar, S. Paterson, R. Santinelli, A.C. Smith, M.S. Miguelez, S.G. Jimenez, DIRAC: a community grid solution. J. Phys.: Conf. Ser. **119**, 062048 (2008)

[143] C. Grefe, S. Poss, A. Sailer, A. Tsaregorodtsev, ILCDIRAC, a DIRAC extension for the linear collider community. J.Phys.Conf.Ser. **513**, 032077 (2013). https://doi.org/10.1088/1742-6596/513/3/032077. CLICdp-Conf-2013-003

[144] O. Viazlo, A. Sailer, Efficient iterative calibration on the grid using iLCDirac. EPJ Web Conf. **245**, 03003 (2020). https://doi.org/10.1051/epjconf/202024503003

[145] L. Valentini, Integration of FCCee workflows in iLCDirac for large scale Monte Carlo productions. Master's thesis, Università deglie Studi di Padova (2024). https://doi.org/20.500.12608/64700. URL https://cds.cern.ch/record/2905533. CERN-THESIS-2024-098