



# interTwin

---

## D7.2 Report on requirements and thematic modules definition for the physics domain

Status: **UNDER EC REVIEW**

Dissemination Level: public



Funded by the  
European Union

**Disclaimer:** Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them


## Abstract

### Key Words

Particle detector simulation, Lattice QCD, Radio Astronomy, Gravitational Waves, High Energy Physics, Digital Twin

interTwin co-designs and implements the prototype of an interdisciplinary Digital Twin Engine (DTE). The developed DTE will be an open source platform that includes software components for modelling and simulation to integrate application-specific Digital Twins. InterTwin WP7 will provide the aforementioned sets of software components, called thematic modules, for the use cases defined in WP4.

The current document consists of a report on the high-level description of the physics domain use cases and of the related designed thematic modules. Additionally, this deliverable includes the set of common requirements derived from the analysis of the use cases and thematic modules that will need to be available in the DTE core modules, designed in WP6.

Document Description			
D7.2 Report on requirements and thematic modules definition for the physics domain			
Work Package number WP7			
Document type	Deliverable		
Document status	UNDER EC REVIEW	Version	1
Dissemination Level	Public		
Copyright Status	 <p>This material by Parties of the interTwin Consortium is licensed under a <a href="https://creativecommons.org/licenses/by/4.0/">Creative Commons Attribution 4.0 International License</a>.</p>		
Lead Partner	CERN		
Document link	<a href="https://documents.egi.eu/document/3956">https://documents.egi.eu/document/3956</a>		
DOI	<a href="https://doi.org/10.5281/zenodo.8036997">https://doi.org/10.5281/zenodo.8036997</a>		
Author(s)	<ul style="list-style-type: none"> <li>• Kalliopi Tsolaki (CERN)</li> <li>• Sofia Vallecorsa (CERN)</li> <li>• David Rousseau (IN2P3)</li> <li>• Isabel Campos (CSIC)</li> <li>• Yurii Pidopryhora (MPG)</li> <li>• Sara Vallero (INFN)</li> <li>• Alberto Gennai (INFN)</li> <li>• Massimiliano Razzano (INFN)</li> </ul>		
Reviewers	<ul style="list-style-type: none"> <li>• Andrea Manzi (EGI)</li> <li>• Daniele Spiga (INFN)</li> </ul>		
Moderated by:	<ul style="list-style-type: none"> <li>• Sjomara Specht (EGI)</li> <li>• Charis Chatzikyriakou (EODC)</li> </ul>		
Approved by	Christoph Reimer (EODC) on behalf of TCB		

Revision History			
Version	Date	Description	Contributors
V0.1	28/02/2023	ToC	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa (CERN)
V0.2	30/03/2023	First phase of contributions in section 2	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa, (CERN) Isabel Campos (CSIC)
V0.3	17/04/2023	Second phase of contributions in sections 2, 3, 4	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa, (CERN) Isabel Campos, (CSIC) Yurii Pidopryhora (MPG)
V0.4	30/04/2023	First draft version	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa, (CERN) David Rousseau, (IN2P3) Isabel Campos, (CSIC) Yurii Pidopryhora, (MPG) Sara Vallero, (INFN) Alberto Gennai (INFN)
V0.5	05/05/2023	Complete draft	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa, (CERN) David Rousseau, (IN2P3) Isabel Campos, (CSIC) Yurii Pidopryhora, (MPG) Sara Vallero, (INFN) Alberto Gennai (INFN)
V0.6	15/05/2023	First version with reviewers comments	Andrea Manzi, (EGI) Daniele Spiga (INFN)
V0.7	25/05/2023	Version Addressing reviewers comments	Kalliopi Tsolaki, (CERN) Sofia Vallecorsa, (CERN)

			David Rousseau, (IN2P3) Isabel Campos, (CSIC) Yurii Pidopryhora, (MPG) Sara Vallero, (INFN) Alberto Gennai, (INFN) Massimiliano Razzano (INFN)
V0.8	02/06/2023	Section 2.1 extended	Isabel Campos (CSIC)
V0.9	09/06/2023	Version ready for TCB approval	Christoph Reimer (EODC)
<b>V1.0</b>	13/06/2023	<b>Final</b>	

## Terminology / Acronyms

Term/Acronym	Definition
GW	Gravitational Wave
QCD	Quantum Chromodynamics
GAN	Generative Adversarial Network
HEP	High Energy Physics
MC	Monte Carlo
HL-LHC	High Luminosity - Large Hadron Collider

Terminology / Acronyms: <https://confluence.egi.eu/display/EGIG>

## Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>9</b>
1.1	Scope .....	9
1.2	Document Structure.....	9
<b>2</b>	<b>Initial design of thematic modules in the physics domain .....</b>	<b>10</b>
2.1	<b>T7.1 Lattice QCD simulations and data management.....</b>	<b>10</b>
2.1.1	Advanced Data management for Lattice QCD .....	10
2.1.2	Generative models using Machine Learning.....	12
2.2	<b>T7.2 Noise simulation for radio astronomy .....</b>	<b>13</b>
2.2.1	Astrophysical analysis of the real data.....	18
2.2.2	Theoretical modelling of the source/telescope system .....	19
2.2.3	Development of a fast and scalable C++ implementation.....	20
2.3	<b>T7.3 GAN-based thematic modules to manage noise simulation, low-latency de-noising and veto generation for Gravitational Waves.....</b>	<b>22</b>
2.3.1	Use-case description .....	22
2.3.2	High-level architecture of the DT implementation .....	23
2.3.3	The Training DT subsystem .....	24
2.3.4	The Inference DT subsystem.....	24
2.4	<b>T7.7 Fast particle detector simulation with GAN .....</b>	<b>27</b>
2.4.1	Use case overview.....	27
2.4.2	3DGAN component implementation .....	30
<b>3</b>	<b>Requirements for the thematic modules in the physics domain .....</b>	<b>34</b>
3.1.1	General Description and Categorization of requirements.....	34
3.2	<b>Storage I/O .....</b>	<b>35</b>
3.2.1	Input data requirements.....	35
3.2.2	Expected data volume.....	35
3.3	<b>Databases.....</b>	<b>35</b>
3.4	<b>Computing.....</b>	<b>36</b>
3.5	<b>OS and execution framework .....</b>	<b>36</b>
3.6	<b>Machine Learning .....</b>	<b>36</b>
3.7	<b>Real-time data acquisition and processing .....</b>	<b>36</b>
3.8	<b>Data formats.....</b>	<b>36</b>
3.9	<b>Software stack .....</b>	<b>36</b>
3.10	<b>Visualisation .....</b>	<b>37</b>



3.11	Data Sharing .....	37
4	Testbed infrastructure .....	38
4.1	Fast particle detector simulation with GAN testbed .....	38
5	Conclusions .....	40
6	References .....	41

## Table of Figures

Figure 1 - Description of the modules involved in the workflow of the simulation .....	11
Figure 2 - C4 graphical representation of the Data Lake architecture needed in Lattice QCD.....	12
Figure 3 - Graphical representation of the classical generation of configurations using Monte Carlo algorithms.....	13
Figure 4 - Graphical representation of the Normalizing Flows method including a correcting accept/reject step to account for the fact that the model cannot be perfectly trained. ....	13
Figure 5 - Examples of the 4 basic time-frame types .....	16
Figure 6 - Distribution of the four main types of time frames in the Effelsberg/Crab pulsar data set. ....	17
Figure 7 - Three parallel interacting subprojects of the task.....	18
Figure 8 - General outline of the pulsar signal digital twin structure. ....	19
Figure 9 - Tentative overview of languages and libraries in the final software product.....	20
Figure 10 - Diagram of the final software product (in the C4 model .....	21
Figure 11 - High-level architecture of the DT.....	23
Figure 12 - System Context diagram (in the C4 model) of the DT for the veto pipeline. ....	23
Figure 13 - Container diagram (in the C4 model) of the Training subsystem.....	25
Figure 14 - Container diagram (in the C4 model) of the Inference subsystem.....	26
Figure 15 - Shower created by primary particle entering the detector volume and interacting with the detector material. ....	28
Figure 16 - Detailed (full) particle simulation with Geant4 (R3) .....	29
Figure 17 - Detailed (full) particle simulation; fast particle simulation using ML techniques (R3) .....	29
Figure 18 - Fast particle detector simulation using ML techniques high level workflow composition (R3).29	
Figure 19 - Fast particle detector simulation using ML techniques high level workflow composition and its connections with other work packages' components in C4 format diagram.....	31
Figure 20 - 3DGAN model architecture .....	38

## **Executive summary**

The present deliverable D7.2 consists of a report on requirements and thematic modules definition for the physics domain Digital Twins applications. It is a collective document written by scientists involved in the development of the physics domain thematic modules. This document describes the different use cases that will derive the aforementioned thematic modules which will be integrated into the Digital Twin Engine (DTE) that the interTwin project is developing. Moreover, an analysis of the specific thematic modules' design and implementation requirements is included. The DTE needs to provide support for use cases' technical requirements concerning the following aspects: I/O storage, databases, operating systems and execution frameworks, machine learning, real-time data acquisition and processing, data formats, software stack, as well as visualisation and data sharing methods. Lastly, already developed proof of concepts are described to showcase thematic modules capabilities and functionalities.



# 1 Introduction

## 1.1 Scope

This document gives an overview of the physics domain thematic modules (T7.1, T7.2, T7.3, T7.7) and their requirements developed by the interTwin project.

The thematic modules will enhance the capabilities of the core engine running the Digital Twins by adding functionalities to several fields, such as:

- Machine Learning (ML) based analysis for QCD simulation configurations and for time series
- Noise signals classification, noise analysis, de-noising, and veto generation
- Generative Adversarial Networks (GAN) based Lattice QCD configurations generation, noise simulation, particle detector simulation
- Particle physics validation techniques capable of assessing different aspects of model performance
- Fast simulation of High Energy Physics detectors

The design of the thematic modules will follow the specific requirements provided by the WP4 – Technical co-design and validation with research communities, where the use cases are being defined. Additionally, specific requirements for each thematic module will be defined and listed in this document.

## 1.2 Document Structure

The document is structured in the following manner:

- **Section 2** includes the description of the initial design of the thematic modules developed for each individual physics use case, starting with the lattice QCD simulations and data management (T7.1), the noise simulation for radio astronomy (T7.2), the GAN-based thematic modules to manage noise simulation, low-latency de-noising and veto generation for Gravitational Waves (T7.3), and closing with the fast particle detector simulation with GAN (T7.7).
- **Section 3** collects the requirements for the thematic modules described in section 2. It is divided into several categories: input and output data requirements, data volumes, databases, computing, OS and execution frameworks, machine learning requirements, real-time data acquisition and processing, data formats, software stack, visualisation, and data sharing.
- **Section 4:** Testbed infrastructure presents frameworks developed as proof of concepts that test the capabilities of the physics domain thematic modules.

- The deliverable ends with **section 5** that draws conclusions from the perspective of the physics use cases: it summarises studies and analyses performed during the first eight months of project's life, that resulted in the first design of the thematic modules and their technical requirements.

## 2 Initial design of thematic modules in the physics domain

### 2.1 T7.1 Lattice QCD simulations and data management

The aim of Lattice QCD is shedding light on the properties of Quantum Chromodynamics in the limit of low energies/strong couplings, where perturbation theory breaks down, and numerical approaches become mandatory. In interTwin the objectives are exploring two use cases addressing the status of Lattice QCD simulations: a classical scenario, with large scale simulations in HPC; and a second scenario, Machine Learning-based simulations, an area under development in the community, at the proof-of-concept level, therefore requiring few resources.

#### 2.1.1 Advanced Data management for Lattice QCD

Simulations are executed at large scale in HPC systems controlled by a batch system (such as slurm<sup>1</sup>). The workflow involves generation of configurations and data analysis, both are computing-intensive tasks. The simulation modules needed are shown in the **Figure 1**;

---

<sup>1</sup> <https://slurm.schedmd.com>

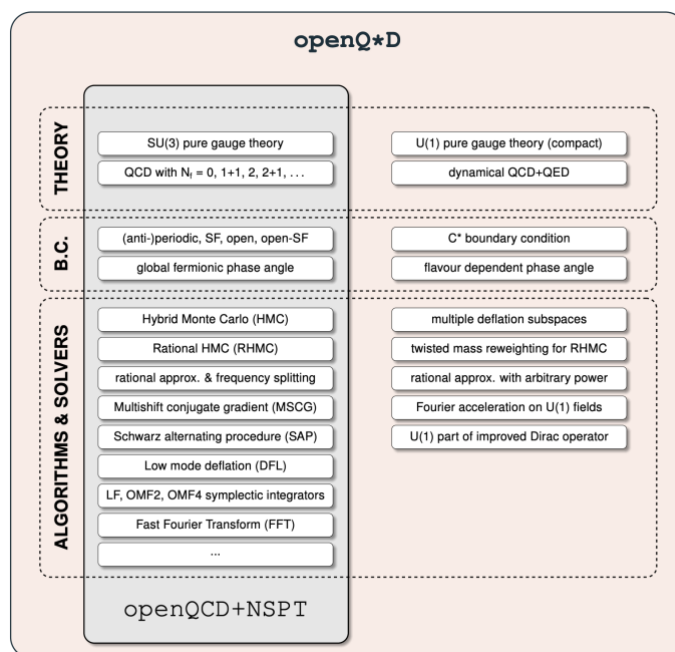


Figure 1 - Description of the modules involved in the workflow of the simulation

The simulation codes are available in Gitlab<sup>2</sup> and described in a paper [R11].

In order to facilitate data analysis, the configurations should be made readily available to the members of the collaboration in a controlled way, for example by using federated identities, and group-based access control. In the most frequent scenario, the members of the collaboration should have group-access enabled to read the data. A few of them, those in charge of generating data, should also have writing access rights. A data sharing model following a Data Lake architecture (WP5) would be desirable.

Two types of runs are executed: generation and measurement runs. Generation takes place in large HPC facilities. Analysis can be off-loaded to smaller facilities, provided the data generated in the HPC system is made available. In both cases, certain files (the gauge configurations, or simply configurations) are regularly written and read to/from disk. The size of a configuration in this example is about 20GB. Usually orders of 1000 configurations are generated. Reading a configuration from disk takes 111s and writing it to disk takes 72s in a modern HPC parallel filesystem such as LUSTRE. For illustration purposes, generating one configuration takes about 7 hours in 4096 cores.

In the measurement runs, a configuration is read from the disc and the desired observable is calculated. In this case, I/O operations constitute 5% of the total run time. Each run also reads an input file at the beginning and writes log files with monitoring information and the result of the calculated observables. These extra I/O operations involve a stream of a few kBytes of data, and their impact on the total run time is completely negligible.

<sup>2</sup> <https://gitlab.com/rcstar/openQxD>

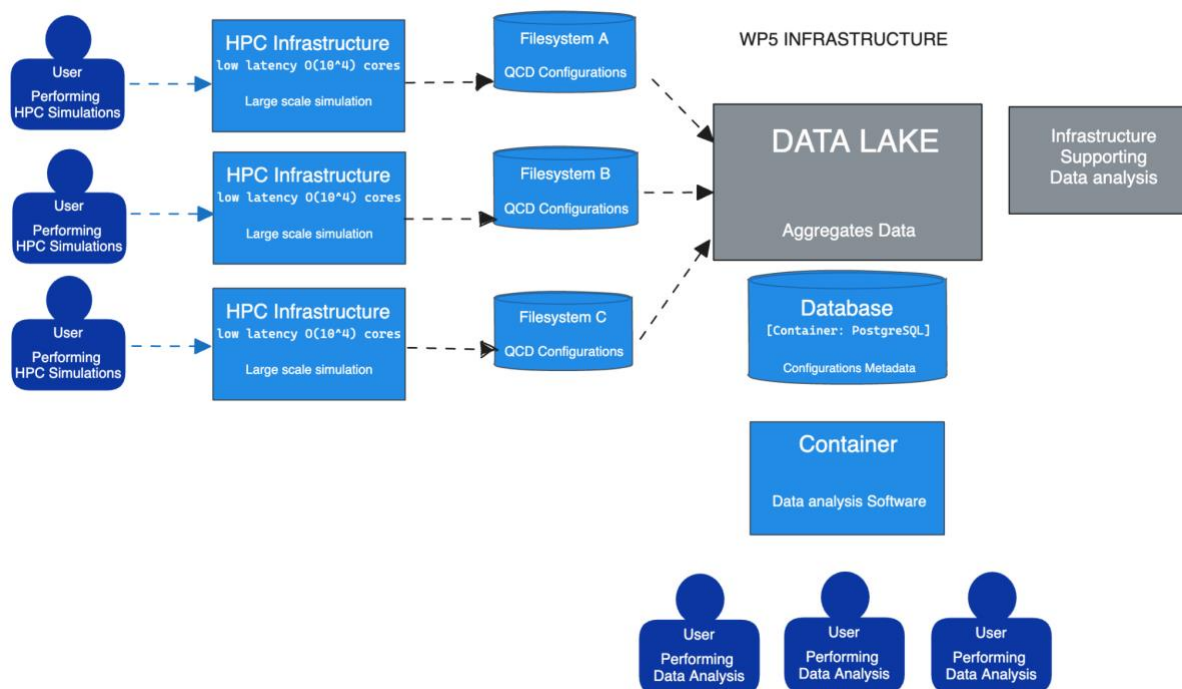


Figure 2 - C4 graphical representation of the Data Lake architecture needed in Lattice QCD

### 2.1.2 Generative models using Machine Learning

The efficiency of general purpose Monte Carlo algorithms decreases dramatically when the simulations need to take place near critical points due to critical slowing down. This is a general phenomenon in simulations in Physics related to phase transitions, which happens as well in Lattice QCD, for example with simulations at very fine distances that are needed for extrapolation to the continuum limit. Simulations need to take place in areas of the parameter space where topology freezing (among other factors) induce very large autocorrelations.

If Machine Learning could help speed-up the field configuration generation in those parts of the parameter space is a subject under investigation. A series of recent studies suggest that using Normalizing Flows (a class of deep generative models) may help to improve this situation (a review is available for instance at [R12] and a block diagram illustrating the method is shown in Figure 2). The underlying idea is using Machine Learning techniques to map the theory of interest to a “simpler” theory, easier to simulate. This approach has the potential to become more efficient than traditional sampling especially when the concept of transfer learning is utilised.

However, the costs associated with the (highly complex) sampling from the path integral, are transferred to the training of a model. The question under investigation is therefore how expensive it is to train a model compared with making a classical Monte Carlo simulation.

Preliminary studies [R13], have demonstrated the proof of concept for simple models in two dimensions. However, further studies indicate that the training cost in CPU time can

be, in general, prohibitively high for large lattices, and the acceptance rates in the accept/reject step (Figure 2) drop fast as the lattice size increases unless better architectures and methods are achieved (see for example [R14] )

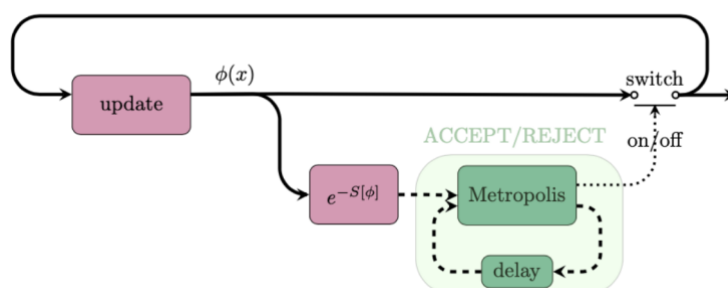


Figure 3 - Graphical representation of the classical generation of configurations using Monte Carlo algorithms

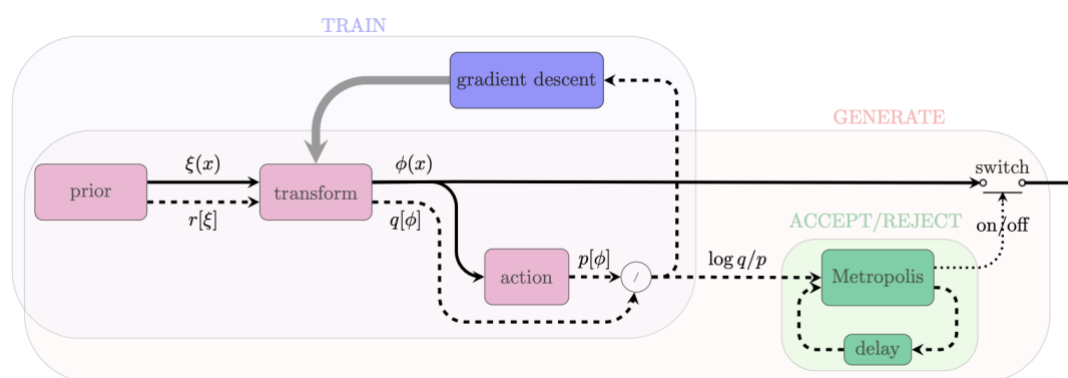


Figure 4 - Graphical representation of the Normalizing Flows method including a correcting accept/reject step to account for the fact that the model cannot be perfectly trained.

The purpose of this work is designing better architectures for Machine Learning models so that the acceptance rates become reasonable (~50% or more) as the volume of the lattice increases. The requirements in terms of resources are not as in the classical Monte Carlo simulation since the methodology is still at the proof of concept level.

A typical Jupyter notebook of the type required for these studies can be found in this reference[R15].

## 2.2 T7.2 Noise simulation for radio astronomy

This task is designed to be instrumental in solving a big problem that is about to arise in modern observational astronomy in general and radio astronomy in particular, and to become one of the largest issues in the whole field: the problem of data overflow. Previous generations of telescopes typically produced no more than a few petabytes of data per year, thus the raw data was generally kept either indefinitely or long enough for the science team to reduce and analyse it, and then approve the deletion, which meant several months or even years. With the arrival of the new so-called Square Kilometre

Array3 "pathfinders", such as South African MeerKAT<sup>4</sup> or Australian ASKAP<sup>5</sup>, the data acquisition rate increases enormously, these tools can easily produce several petabytes of raw data per week<sup>6</sup>. No current astronomical institution can handle keeping such volumes of data even for a month or employ a team of experts large enough to quickly process it or sort through it manually. Thus, it is crucial to develop automated decision-making systems that can sort through the raw data in real or near-real time (since telescopes usually have downtime due to maintenance or source availability, the data can be pooled for short periods of time of order of days) and separate the data flow into the scientifically important data that must be kept and the rest that can be safely deleted.

Another reason to be able to automatically sort through the incoming data is that modern radio astronomy is increasingly interested in transient sources. Previously sources had to be observed for long periods of time to be able to achieve the necessary signal to noise ratio, thus it was possible to observe reliably or even discover at all only permanent or fast periodic<sup>7</sup> sources like pulsars. Since the new telescopes are much more sensitive, they can systematically probe the transient radio sky, which currently is generally unknown. Such studies are very important, since it is believed that the transients<sup>8</sup> result from very far and enormously energetic exotic events (like black hole collisions) that may provide essential clues for the areas of physics that cannot be studied experimentally in any other way, e.g., quantum gravity. An automated expert system can help with this: if something like a transient source (or unusual in general) signature is found in the data flow, it can immediately trigger the "target of opportunity" mode of observation for the detected anomaly, and alert the scientists on duty, who would decide the best course of further action. This will also allow us to easily organise concerted efforts of observing rare important sources by a number of instruments, covering a range of wavelengths, e.g., combining Earth-based radio observations with space-based optical and X-ray

---

<sup>3</sup> SKAO: <https://www.skao.int/en>

<sup>4</sup> MeerKAT Radio Telescope: <https://www.sarao.ac.za/gallery/meerkat/>

<sup>5</sup> ASKAP-radio telescope: <https://www.csiro.au/en/about/facilities-collections/atnf/askap-radio-telescope>

<sup>6</sup> Predicted data rate for an SKA pathfinder like MeerKAT is of order 10 Gbytes/s or up to 1 Pbytes/day. The SKA itself is expected to produce up to 200 Pbytes/day, which is ~70 exabytes per year. To put it into perspective, the latter is about the same as the expected data rate of CERN's LHC after the High-Luminosity upgrade (60 exabytes per year) and at about the same time (SKA's first light is expected in 2027 and the High-Luminosity LHC should go online in 2029).

<sup>7</sup> Known pulsars have periods from a few milliseconds to 8 seconds, thus over a typical observational session of several hours one can observe many pulses, which makes pulsars relatively easy to detect and observe. However, if we imagine a transient phenomenon similar to a pulse of a pulsar, but either non-periodic or with periods of order of hours or days, discovering it is close to impossible except by sheer luck.

<sup>8</sup> Examples of such transients that attract a lot of attention in the radio astronomical community are "fast radio bursts" (FRBs), see e.g., [arXiv: 2107.10113](https://arxiv.org/abs/2107.10113) and references thereof.

observations — it is already done today, but with typical response times very far from ideal<sup>9</sup>.

The third reason for this task is that current common radio astronomy software tools are inadequate, they are computationally slow and handle parallelization poorly. For the tasks at hand, we would like to build tools that can be efficiently run on modern HPC clusters, with scalability to at least hundreds of cores. It is connected to the main task of the ML data classification system in way that, although the classification system itself will be run on ordinary observatory computers embedded in a telescope's data acquisition system, the training of new models before each new type of observation, which is the most computationally intensive task, will probably have to be performed on supercomputers.

Of course, to be able to detect special and important events in the data, one has first to understand the regular and mundane features of the data stream well. In radio astronomy this primarily means noise and radio-frequency interference (RFI).

Our starting point for this task is building a ML-based data-labelling system that reads the data flow coming from a real telescope observing a pulsar. Pulsars are ideal test subjects for this task since they reliably produce periodic bursts of scientifically significant data with certain variability in signal strength and other parameters.

However, because of the nature of the pulsars, objects that are "silent" most of the time, a telescope observing a pulsar mostly records either an "empty" data stream, i.e., only the noise, or some sort of radio-frequency interference (RFI) due to artificial or natural electro-magnetic phenomena unrelated to space. We separate two main types of RFI: narrow-band RFI (NBRFI) that is present only on some frequencies of the observation band and broad-band RFI (BBRFI) that covers the whole observation band. Examples of the 4 basic time-frame types are shown in **Figure 5**. Thus, the basic task of the ML classifier is to label each small fragment of the data stream (a time frame) as one of the standard categories: signal, none (noise), NBRFI and BBRFI. In reality a few extra labels need to be introduced: "other" (something that cannot be classified into known categories) and mixed ones (e.g., signal with NBRFI, or NBRFI with short bursts of BBRFI).

---

<sup>9</sup> Even in the best case scenario when a special "target of opportunity" (ToO) event is expected, and a change of scheduling is proposed in advance for all the observatories involved, the actual triggering of such an event is a complicated and disruptive procedure involving many exchanges between various personnel of many institutions, thus the response time is rarely shorter than a day. Using an automated decision making system with pre-approved criteria can change this to minutes, most of the time taken to actually reposition the telescopes.



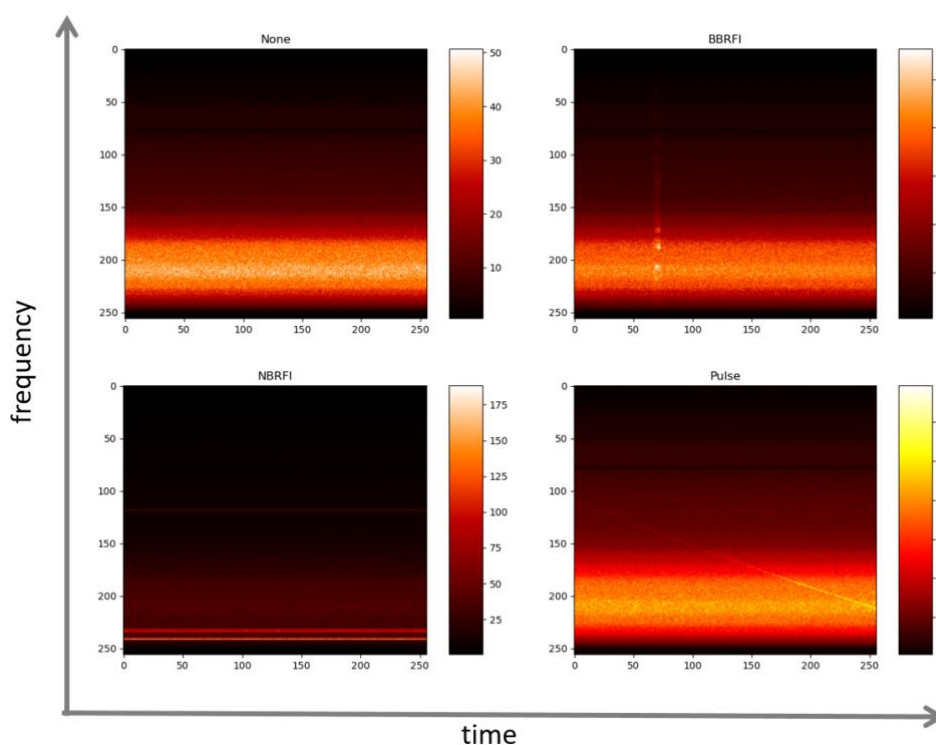


Figure 5 - Examples of the 4 basic time-frame types

As shown in **Figure 5**, the examples of four main types of data “time frames” in a pulsar-observation data set, visualised as 256x256 images. The vertical axis corresponds to frequency, horizontal to time, the value of each point is signal intensity. The top left frame is the most common one, “none” or “empty”, it contains only noise, amplified differently because the telescope’s sensitivity is different for different frequencies, leading to apparent horizontal banding; the top right and bottom left frames contain broad-band (BB) and narrow-band (NB) radio-frequency interference (RFI) represented by brighter vertical or horizontal stripes, these types of data are undesirable but often recorded because the telescope is sensitive to various artificial or natural electro-magnetic phenomena (radio transmissions, emissions by various devices, electric storms etc.). Finally, the bottom-right frame contains a pulsar’s pulse, represented by the diagonal slightly curved line; this is the only desirable type of data frame that we would like to separate from all the other types.

Our first test data set is about 20 minutes of data collected by the Effelsberg 100m radio telescope<sup>10</sup> observing one of the brightest and well-studied pulsars, the Crab pulsar. The size of the dataset is ~12.2 Gb. The set is broken into more than 50,000 time frames, each with 256 spectra in it. These frames were looked through and labelled by hand, so the labelled set can be used for ML training or quality assessment. In the future we will use much longer data sets (up to 100 TB), of different sources and produced by other telescopes. In particular we have already arranged to get data from one of the SKA

<sup>10</sup> Radio Telescope Effelsberg: <https://www.mpifr-bonn.mpg.de/en/effelsberg>



pathfinders, MeerKAT<sup>11</sup>, which will be the final testbed for this task and, as we hope, will ultimately use our software or its descendants in its actual day-to-day operation.

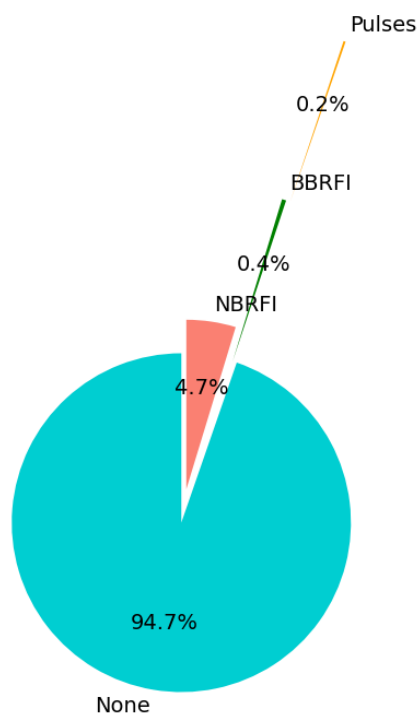


Figure 6 - Distribution of the four main types of time frames in the Effelsberg/Crab pulsar data set.

As shown in **Figure 6**, the Effelsberg/Crab pulsar data set has 94.7% of "none" frames, 4.7% NBRFI, 0.4% BBRFI and only 0.2% of signal frames (there are only a few mixed-type or unidentifiable time frames, so their percentage is negligible and they are not included in this distribution). This illustrates the whole concept well: only 0.2% of the data is scientifically significant, everything else can be safely deleted right as it is being observed, with only some basic info (like time duration of empty periods) kept.

The work is split into three parallel and interacting subprojects (**Figure 7**) which are described in the following sections.

---

<sup>11</sup> The data will come from an ongoing scientific project led by MPIfR radio astronomers. Despite the embargo applicable, we expect hardly any limitations, since we work with low-level unprocessed partial data that has little scientific value. However, this is going to place certain restrictions on access to the original data sets outside of the MPIfR.

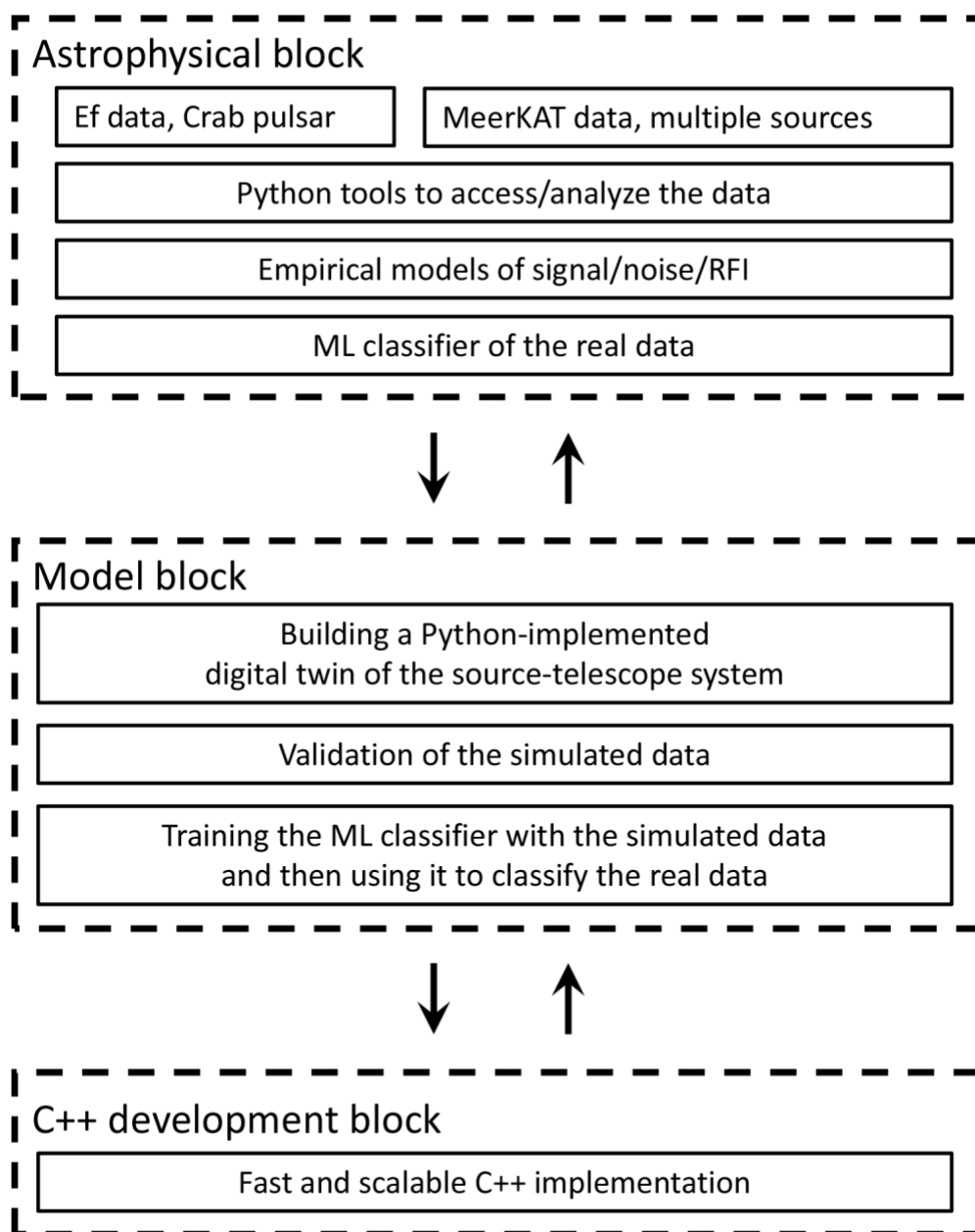


Figure 7 - Three parallel interacting subprojects of the task

### 2.2.1 Astrophysical analysis of the real data.

First of all, we need to understand the data we already have. That means not just labelling, but their detailed signal, noise and RFI properties. And a ML tool is being built to reliably label the dataflow: a convolutional neural network (CNN) based classifier, implemented with TensorFlow in Python, that can label the data with 90% or better accuracy, and is

constantly improved. In the process of creating and improving this tool we study both the data and the ML architecture<sup>12</sup> and implementation strategy.

## 2.2.2 Theoretical modelling of the source/telescope system

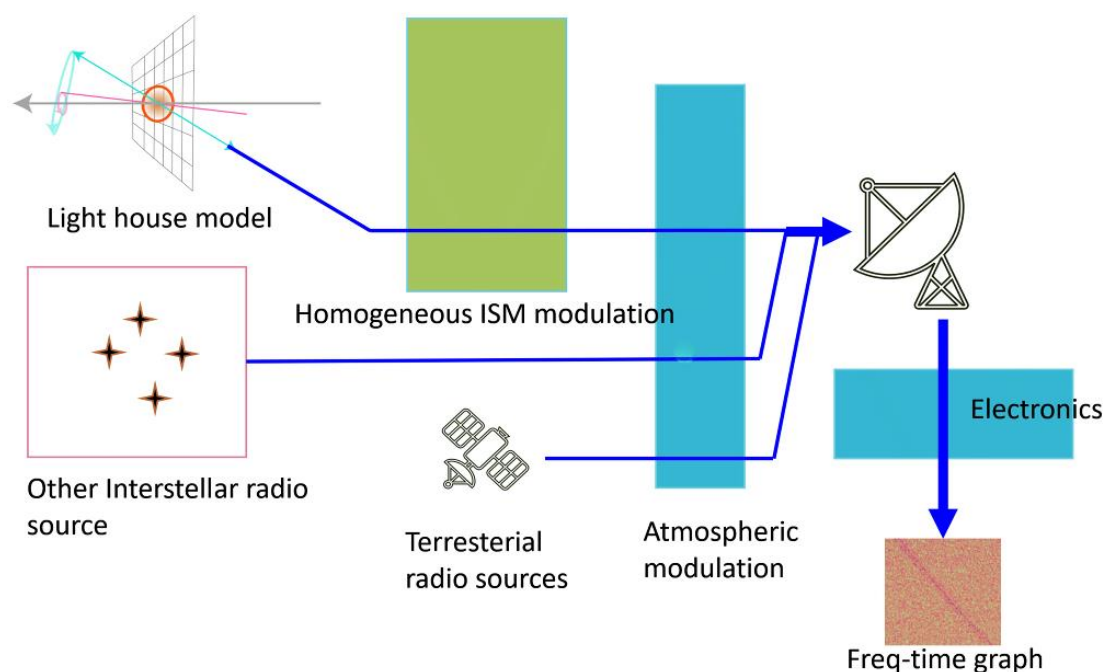


Figure 8 - General outline of the pulsar signal digital twin structure.

This is the main digital twin creation subproject. In it we build a sophisticated model of the pulsar signal, anything that interferes with it and the way it is finally recorded by a telescope. This model will produce data that ideally is indistinguishable from a real telescope observing a real source, with a number of parameters that can be adjusted. The model (**Figure 8**) starts with the simple "lighthouse" representation of a pulsar, but then adds to it interstellar matter (ISM) effects, other cosmic sources, influence of the Earth atmosphere, terrestrial sources, and effects of the telescope's receiving and recording equipment.

Because a lot both in the modelling and in the astrophysical block is highly experimental and based on trial and error, everything in these two blocks is first implemented in Python, for the reason of transparency, clarity, and ease of modification.

<sup>12</sup> Typical questions we are trying to answer at this stage are: what are the best parameters and architecture of the CNN network? What preprocessing of the data (e.g., filtering or averaging) is helpful for optimal handling by the network? Can any other ML algorithms (e.g., SVM) deal better with the same classification task?

### 2.2.3 Development of a fast and scalable C++ implementation.

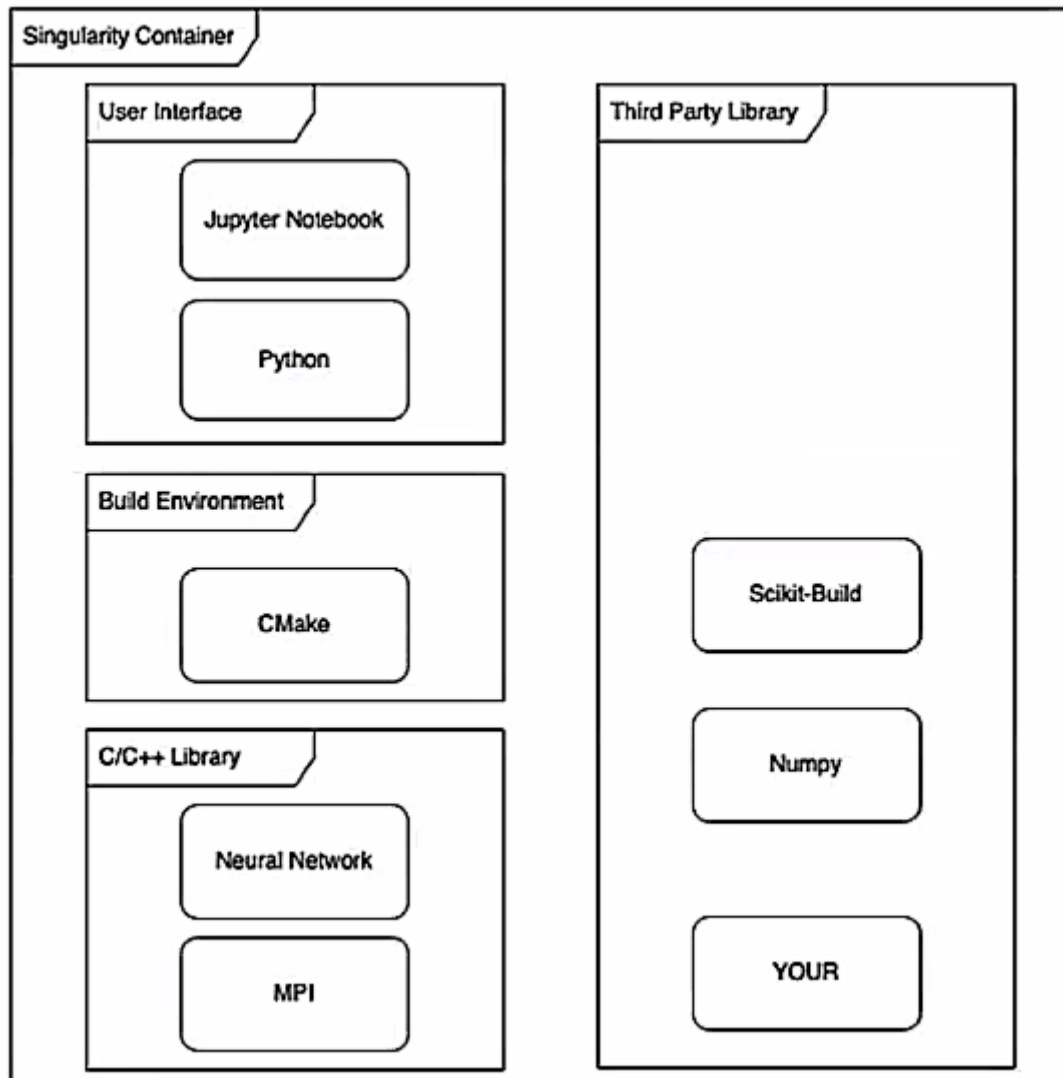


Figure 9 - Tentative overview of languages and libraries in the final software product.

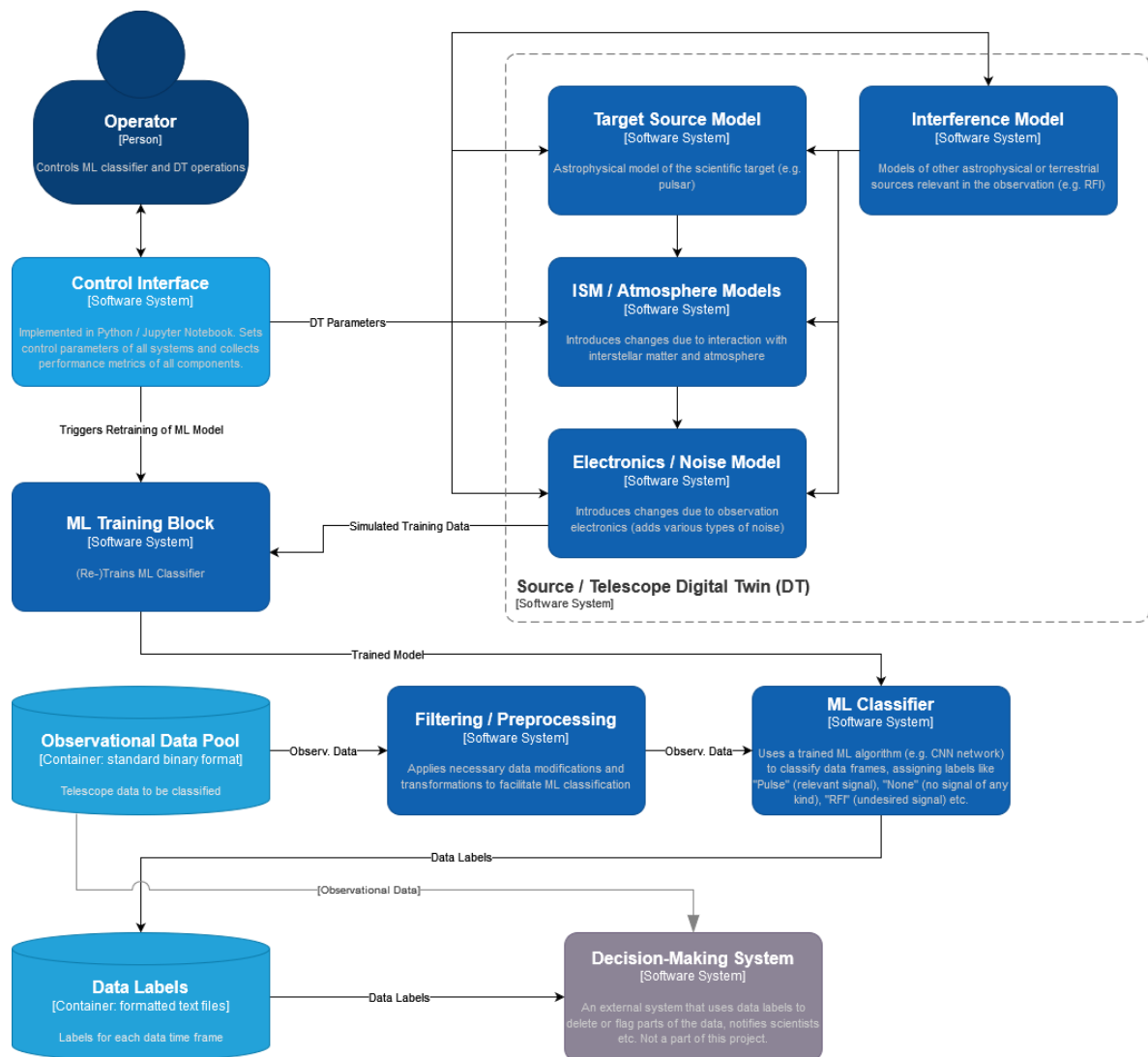


Figure 10 - Diagram of the final software product (in the C4 model)

The stable versions of the ML classifier and the source/telescope digital twin of the activities described in sections 2.2.2 and 2.2.3 are then being rewritten in C++, with speed and scalability being a priority. The software (Figure 9 and Figure 10) has a Jupyter notebook / Python based user interface. All of it is to be included in a Singularity container for easy distribution to any platform.

The purpose of the digital twin development is twofold. On one hand we would like to have an endless supply of training/testing material for the ML-based classifier with parameters that can be adjusted for a particular observation. On the other hand, building a digital twin of a telescope signal with known parameters can help with identification of the real signals in the telescope's data flow. For example, at the end we would like to be able to classify the scientifically significant part of the data not just as a "signal", but "a pulse of a pulsar with such and such physical parameters".

The final version of the classifier is intended for use with the real data flow of the MeerKAT telescope, and, later, possibly with other telescopes as well. But these goals are already beyond the scope of the current project.

## 2.3 T7.3 GAN-based thematic modules to manage noise simulation, low-latency de-noising and veto generation for Gravitational Waves

### 2.3.1 Use-case description

The sensitivity of Gravitational Wave (GW) interferometers is limited by noise. We will use Generative Adversarial Networks (GANs) to produce a Digital Twin (DT) of the Virgo interferometer to realistically simulate transient noise in the detector. In the first phase, we will use the GAN-based DT to generate synthetic strain data. Strain is the channel that measures the deformation induced by the passage of a gravitational wave. Furthermore, the detector is equipped with sensors that monitor the status of the detector's subsystems as well as the environmental conditions (wind, temperature, seismic motions) and whose output is saved in the so-called auxiliary channels. In a second phase, also in the perspective of the Einstein Telescope, we will use the trained model to characterise the noise and optimise the use of auxiliary channels in vetoing and denoising the signal in low-latency searches, i.e., those data analysis pipelines that search for transient astrophysical signals in almost real time. This will allow the low-latency searches (not part of the DT) to send out more reliable triggers to observatories for multi-messenger astronomy.

**Figure 11** shows the high-level architecture of the DT. Data streams from auxiliary channels are used to find the transfer function of the system producing non-linear noise in the detector output. The output function compares the simulated and the real signals in order to issue a veto decision (to further process incoming data in low-latency searches) or to remove the noise contribution from the real signal (denoising).

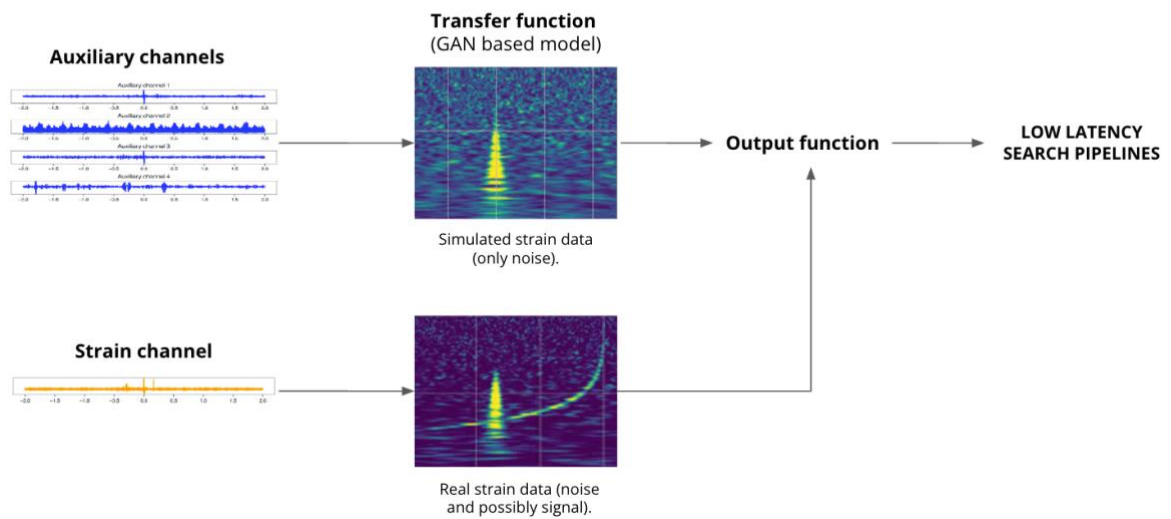


Figure 11 - High-level architecture of the DT

### 2.3.2 High-level architecture of the DT implementation

**Figure 12** shows the System Context diagram (in the C4 model) of the DT for the veto pipeline. In the rest of the document, we will focus on the veto pipeline only, but similar diagrams also apply to the denoising pipeline.

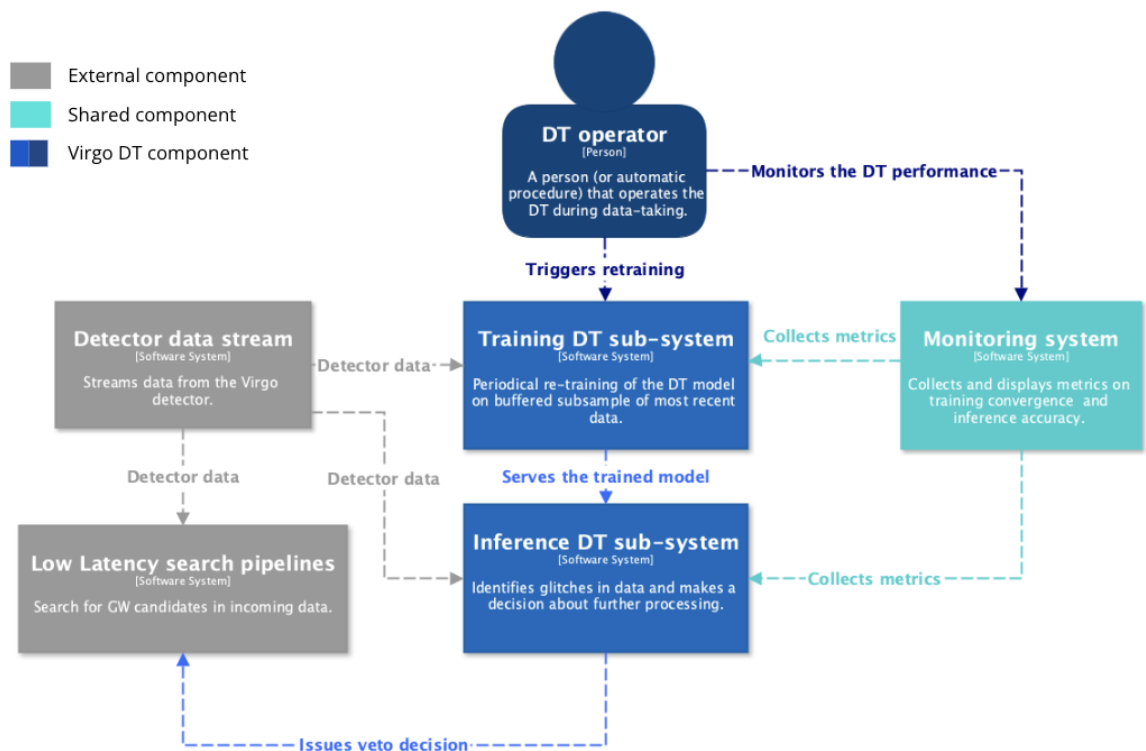


Figure 12 - System Context diagram (in the C4 model) of the DT for the veto pipeline.

Two main subsystems characterise the DT architecture: the *Training DT subsystem* and the *Inference DT subsystem*. The Training subsystem is responsible for the periodical re-training of the DT model on a buffered (on disk) subsample of most recent data. The Inference subsystem processes a stream of incoming data, identifies glitches, and issues a decision about further processing (veto).

The above subsystems are operated by a *DT Operator*, which could be a person or an automated procedure (in a later stage) that operates the DT during data-taking. The *DT Operator* monitors the operations of the DT by leveraging a *Monitoring System* that collects and displays metrics on training convergence and inference accuracy. The *Monitoring System* is not specific to this use-case and it's a part of the core components developed by WP6.

### 2.3.3 The Training DT subsystem

**Figure 13** shows the Container diagram (in the C4 model) of the Training subsystem. The main components of the subsystem are:

- The *Training application*, based on Pytorch (or Tensorflow), which is responsible for training the GAN model on most recent data. This component needs to be interfaced to the *DT Operator* and the *Monitoring system*.
- The *Preprocessing API*, based on Python and proprietary IGWN libraries, which prepares the input data in a format suited for the Training application.
- The *Data Store API*, based on Python and proprietary IGWN libraries, which converts the Kafka<sup>13</sup> stream of detector data to files.
- The *Data buffer*, most likely a POSIX filesystem, which stores a buffer of most recent detector data to train the GAN model.

The flow of detector data at various processing steps (grey arrows) is also shown in the figure, as well as the flow of DT artefacts (blue arrows). In this subsystem, the only produced artefact is the trained model.

### 2.3.4 The Inference DT subsystem

**Figure 14** shows the Container diagram (in the C4 model) of the Inference subsystem. The main components of the subsystem are:

- The *Simulation application*, based on Pytorch (or Tensorflow), which is the GAN inference application that simulates the strain channel starting from the auxiliary channels data. This component needs to be interfaced to the *Monitoring system*.

---

<sup>13</sup> <https://kafka.apache.org/>



- The *Preprocessing API*, based on Python and proprietary IGWN libraries, which prepares the input data in a format suited for the Inference application. It can be the same module as the one belonging to the *Training subsystem*.
- The *Analysis application*, based on Python and proprietary IGWN libraries, calculates the probability for the strain data to contain a glitch and/or a signal. It needs to receive as input both the simulated and the real strain data.
- The *Veto API*, based on Python and proprietary IGWN libraries, which is an interface to the Virgo Low Latency framework.

The flow of detector data at various processing steps (grey arrows) is also shown in the figure, as well as the flow of DT artefacts (blue arrows). In this subsystem, the DT artefacts are the trained model from the *Training subsystem*, the simulated strain data, a veto decision in a data representation internal to the DT system and a veto decision in a data representation compatible with the Virgo Low Latency framework.

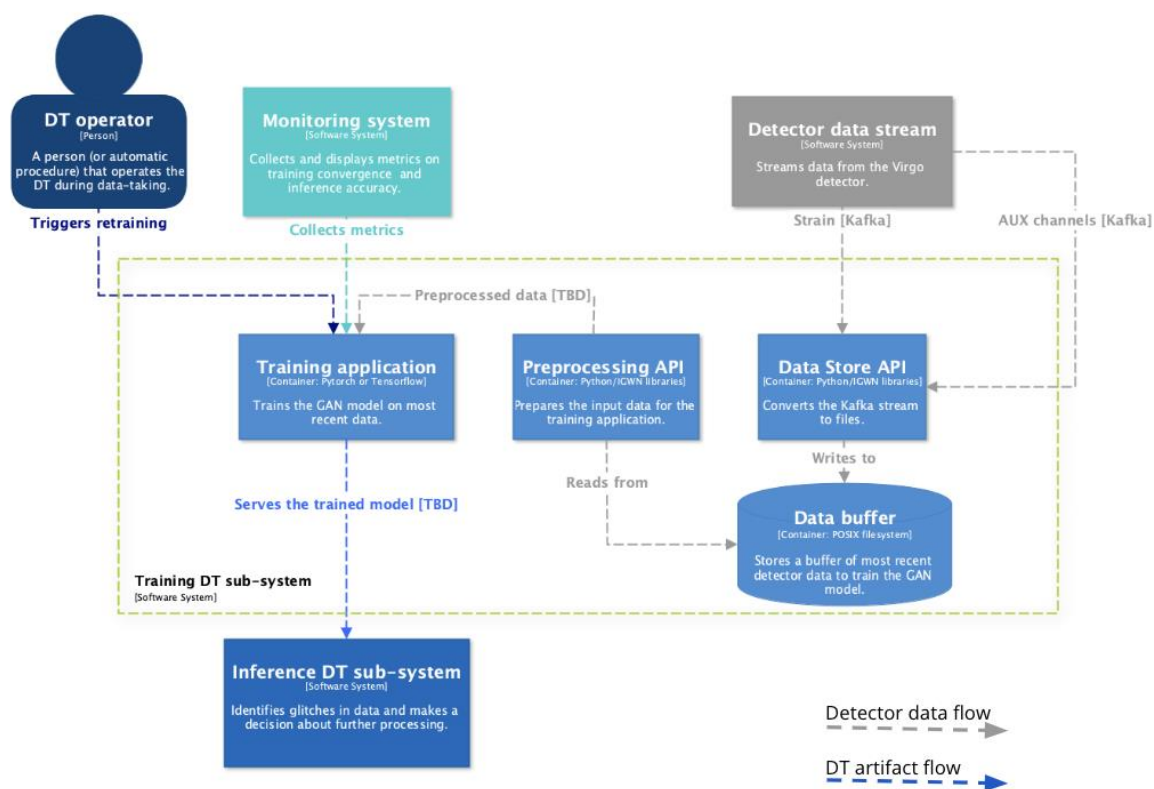


Figure 13 - Container diagram (in the C4 model) of the Training subsystem.

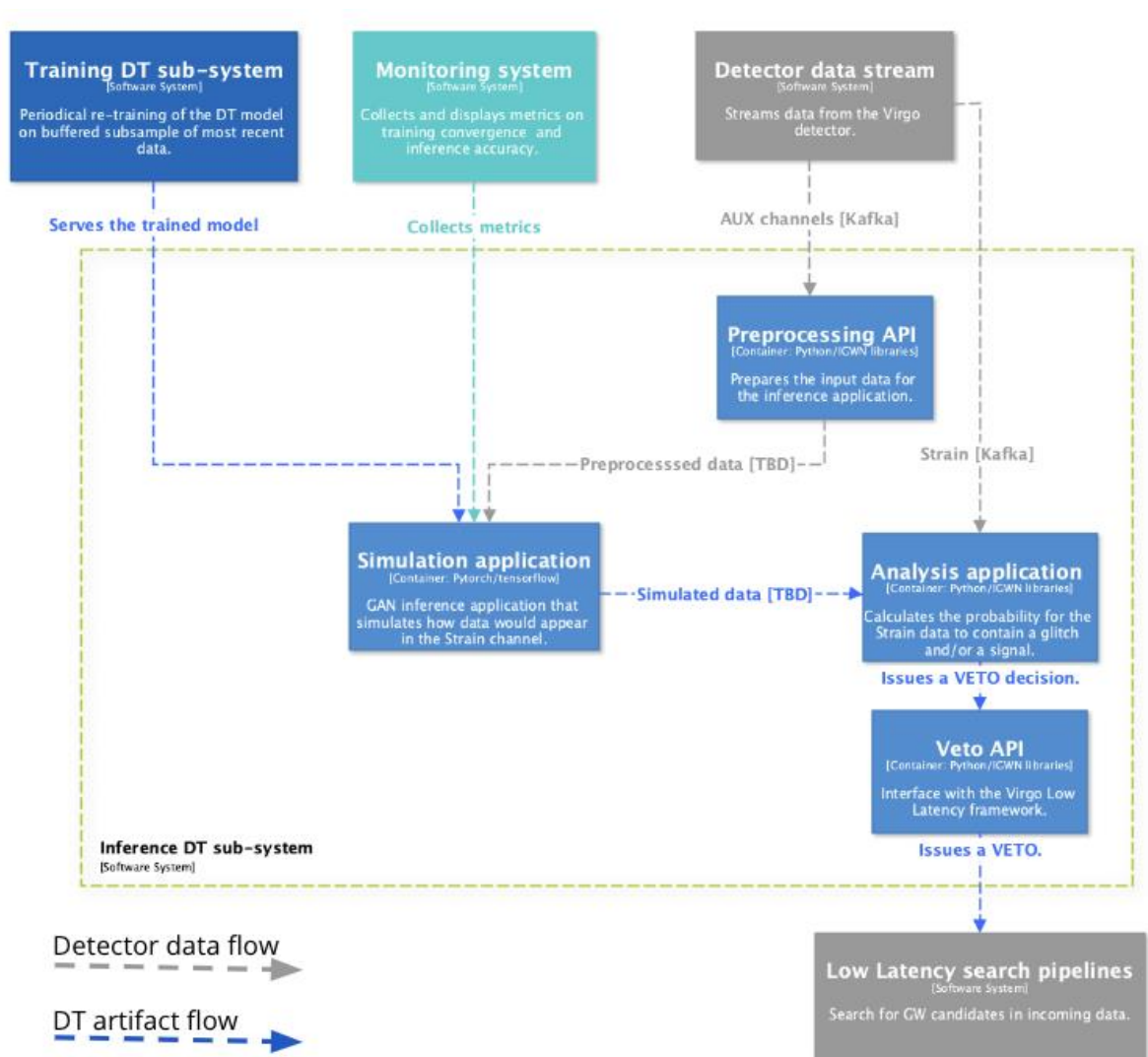


Figure 14 - Container diagram (in the C4 model) of the Inference subsystem.

## 2.4 T7.7 Fast particle detector simulation with GAN

The two components present in the thematic module of T7.7 are:

- the simulation component that incorporates the Monte Carlo-based simulation framework ([R9](#))
- the deep learning (3D Generative Adversarial Network) component ([R2](#)), which will produce deep learning models based on a specified particle detector set up.

These models are integrated and can be run during the simulation step.

More specifically, the objectives of T7.7 are summarised below:

- Optimising the Generative Adversarial Network (GAN)-based model developed for a selected set of [detector geometries](#).
- Integrating WP6 tools for distributed training and hyperparameter optimization.
- Implementing validation techniques capable of assessing different performance aspects, such as accuracy and comparison to classical Monte Carlo, uncertainty estimation, coverage of the support space. This activity will be contributing to the development of an agreed validation standard among the HEP community.

### 2.4.1 Use case overview

Particle detectors measure different particle properties at colliders such as the Large Hadron Collider (LHC). More specifically, the detectors called calorimeters are key components of the whole experimental setup, which are responsible for measuring the energy of the particles. In a collider, the emerging particles travel through the detector and interact with the detector material through the fundamental forces. In particular, within electromagnetic or hadronic calorimeters, showers of secondary particles are created due to the interaction of each new particle with the dense calorimeter material ([Figure 15](#)).

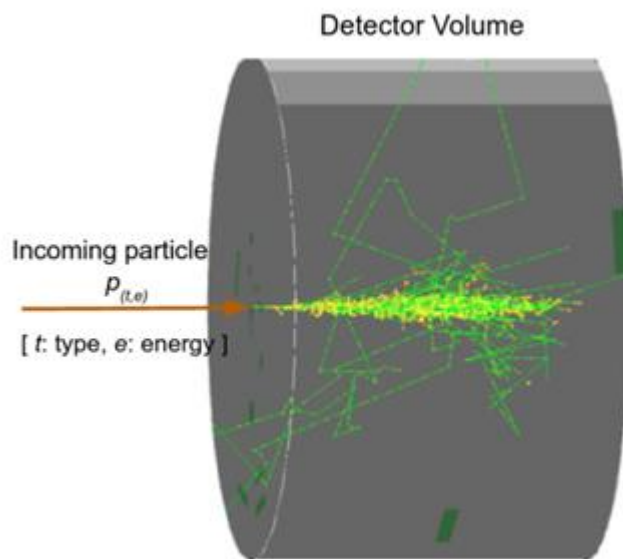


Figure 15 - Shower created by primary particle entering the detector volume and interacting with the detector material.

The secondary particle creation process is inherently a complex stochastic process, and it is typically modelled using Monte Carlo (MC) techniques. These simulations have a crucial role in High Energy Physics (HEP) experiments, and at the same time are very resource-intensive from a computing perspective. Recent estimates show that the HEP community devotes more than 50% of the WLCG computing Grid<sup>14</sup> (which has a total of 1.4 million CPU-cores running 24/7/365 worldwide) to simulation-related tasks (R10). Moreover, Monte Carlo simulations are constrained by the need for accuracy, which will further increase, in the near future with the High Luminosity upgrade of the LHC (HL-LHC<sup>15</sup>). HL-LHC will increase the demand in terms of simulations, and consequently the need for computing resources, as well as the complexity of the associated detector data (R3).

Detector simulation allows scientists to design detectors and perform physics analyses. The simulation toolkit that has been developed and performs particle physics simulations based on MC methods, and is also used in our use case, is Geant4. It provides a highly flexible simulation framework in C++. Moreover, Geant4 is used by large scale experiments and projects from the domains of nuclear medicine, astrophysics, and HEP. It constitutes a set of components which include, geometry and tracking descriptions, detector response modelling, event management, user interfaces and much more.

Given the expected HL-LHC requirements in terms of simulation, the community has long since started developing faster alternatives to Monte Carlo, including deep learning based techniques (R4, R5, R6).

<sup>14</sup> WLCG computing Grid : <https://home.cern/science/computing/grid>

<sup>15</sup> HL-LHC: <https://hilumilhc.web.cern.ch/>

In the calorimeter case, deep learning based fast simulation directly generates the detector output, without reproducing, step by step, each single particle that interacts with the detector material. More specifically, generative models have been used in related HEP applications, as they are able to combine deep learning with statistical inference and probabilistic modelling. A generative model's goal is to learn how to generate data based on a true, unknown distribution describing a finite number of observations. There are several variants of generative models found in literature with the most well-known ones being the Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) and Autoregressive Models ([R1](#), [R7](#), [R8](#)). The thematic module under task 7.7 designed for CERN's use case concerns a fast particle detector simulation paradigm using GANs.

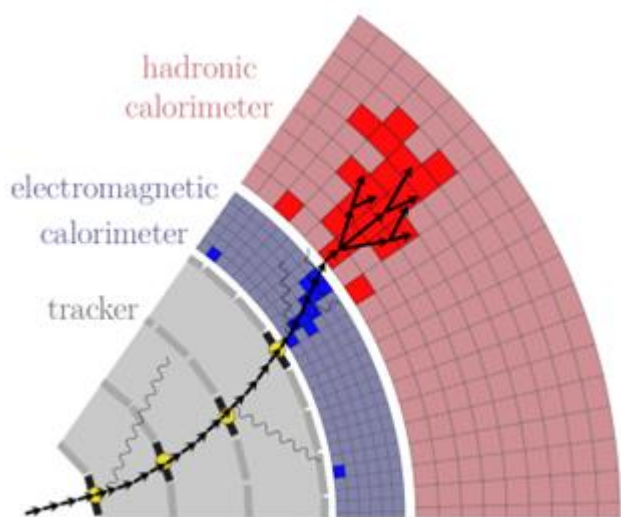


Figure 16 - Detailed (full) particle simulation with Geant4 (R3)

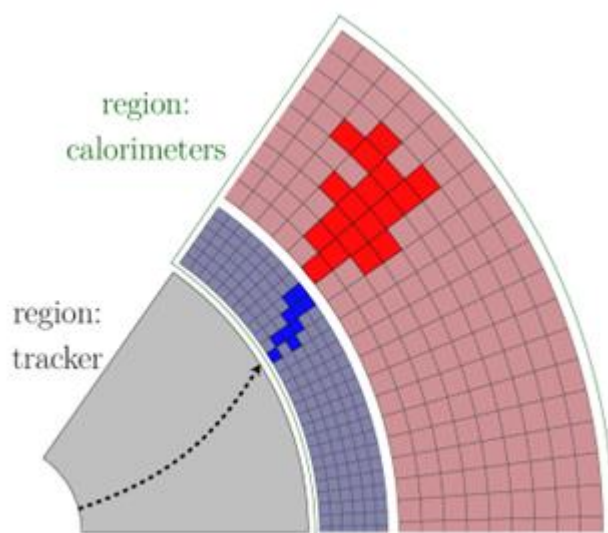


Figure 17 - Detailed (full) particle simulation; fast particle simulation using ML techniques (R3)

Compared to other generative approaches, the Generative Adversarial Network approach is able to demonstrate highly realistic and sharp images ([R1](#)). A GAN can learn a distribution implicitly as it doesn't rely on the explicit computation of probability densities. This use case uses a convolutional GAN, 3DGAN, as calorimeter detectors can be regarded as huge cameras taking pictures of particle interactions ([R2](#)). The voxels (3D calorimeter cells) are generated as monochromatic pixelated images with the pixel intensities representing the cell energy depositions.

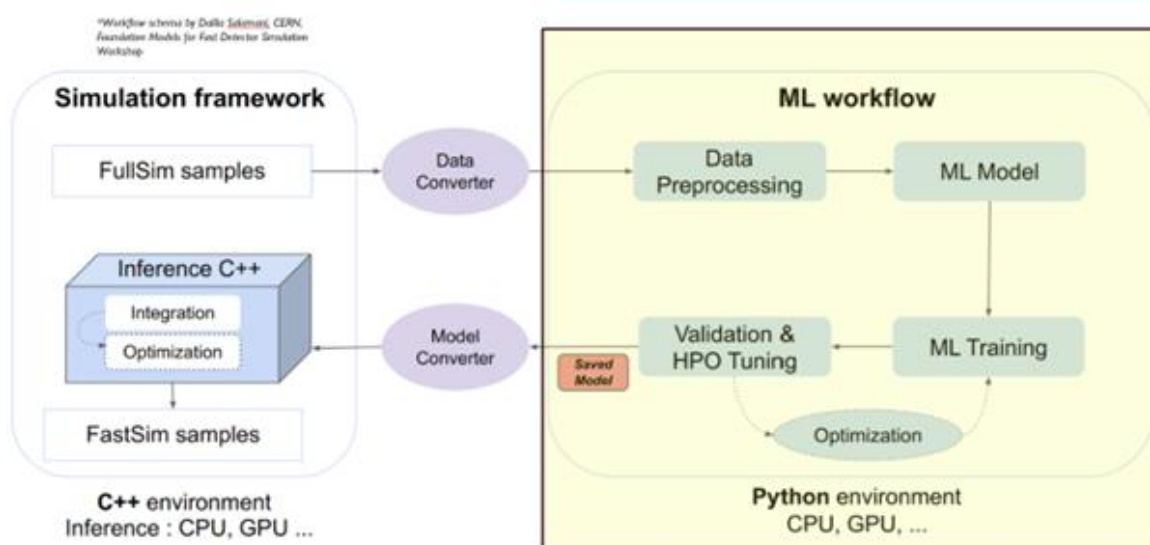


Figure 18 - Fast particle detector simulation using ML techniques high level workflow composition (R3)

### 2.4.2 3DGAN component implementation

An implementation of the 3DGAN approach has been developed and a more detailed description follows. Under [section 4](#) of this document the implementation testbed is referenced.

The 3DGAN architecture can be seen in [Figure 20](#). The generator network implements stochasticity through a latent vector drawn from a Gaussian distribution. The generator input includes the primary particle's initial energy and the angle that it entered the detector, concatenated to the latent vector. The generator network then maps the input to a layer of linear neurons followed by 3D convolutional layers. The discriminator input is an image while the network has only 3D convolutional layers. Batch normalisation is performed between the layers and the LeakyRelu<sup>16</sup> activation function is used for the discriminator layers while the Relu<sup>13</sup> activation function is used for the generator layers. The model's loss function is the weighted sum of individual losses concerning the discriminator outputs and domain-related constraints, which are essential to achieve high level agreement over the very large dynamic range of the image pixel intensity distribution in a HEP task.

The training of this model was inspired by the concept of transfer learning. Meaning that the 3DGAN was trained first for images in a limited energy range and after the GAN converged, the same trained model was further trained with the data from the whole available energy range. The first training step was run for 130 epochs and the second step was run for 30 epochs, both runs utilised GPU infrastructure.

<sup>16</sup> [https://en.wikipedia.org/wiki/Rectifier\\_\(neural\\_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))



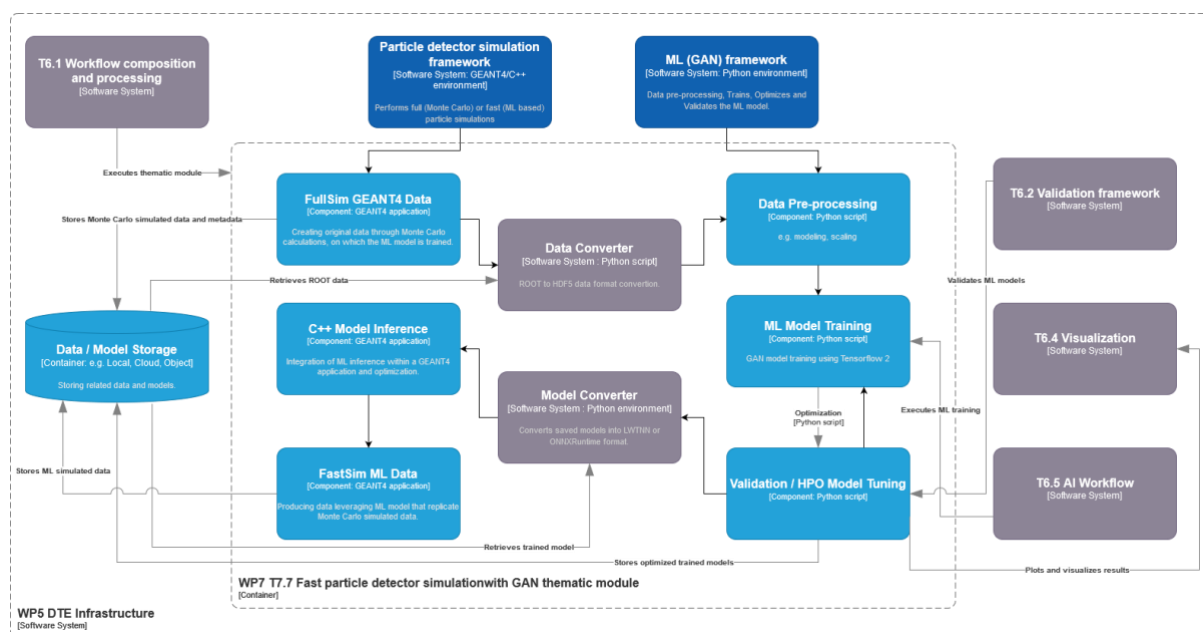


Figure 19 - Fast particle detector simulation using ML techniques high level workflow composition and its connections with other work packages' components in C4 format diagram

Figure 18 shows our digital twin application's workflow, as well as the different components it consists of. Moreover, in Figure 19 the relationships with other work packages and our thematic module are depicted. The ML workflow that includes the 3DGAN model consists of several other modules, the data pre-processing module, the model definition and training module, the validation and hyperparameter optimization module.

- The preprocessing module is responsible for preparing (cleaning, scaling, etc.) and converting into a suitable format (HDF5 format) the simulated data created by GEANT4 (ROOT format). It also encodes the input information such as the calorimeter's geometry identifier, the energy of the primary particle initiating the shower, the angle at which the particle enters the detector, and also its type and/or initial position.
- The preprocessed data are then passed to the GAN model (currently developed using Tensorflow v2<sup>17</sup> and re-implemented in PyTorch<sup>18</sup> in the future) for training.
- The hyperparameter optimization (HPO) tuning module is used for searching for the best set of model hyperparameters (e.g., AutoML<sup>19</sup>, Optuna<sup>20</sup> etc.).

<sup>17</sup> Tensorflow: <https://www.tensorflow.org/>

<sup>18</sup> PyTorch: <https://pytorch.org/>

<sup>19</sup> AutoML: <https://www.automl.org/automl/>

<sup>20</sup> Optuna: <https://optuna.org/>

- The validation module verifies the performance through a set of physics-motivated steps, both at single image quality level and at the sample level.
- Finally, the model is converted into ONNX<sup>21</sup> format and used for inference within GEANT4 (C++ based).

Describing in more detail the above processes and the involved components will help us uncover the technical requirements concerning our digital twin particle detector application.

The dataset used for studying and developing the 3DGAN model (**R2**) (**public dataset**) consists of calorimeter 3D images/arrays of energy depositions with shape 51x51x25, which represent the particle showers. These images were created from simulations performed with GEANT4 software. The output of the Geant4 simulation is ROOT<sup>22</sup> files, which need to be converted into a ML-friendly format HDF5 in order to train the model. The datasets and the converted data (ROOT to HDF5) need to reside in cloud file or object storage, as the volumes do not exceed several 100s of GB. The pre-processing of the data (cleaning, scaling, encoding etc.) is being performed in memory.

During pre-processing, simulation inputs are defined and encoded, i.e., the detector geometry, the energy and angle of the incoming particle. The 3DGAN training requires multiple GPU access and the best model weights need to be saved in a model registry repository, in order to be available during inference.

The performance of the model is evaluated in the validation module through the creation of histograms describing particle shower observables. Shower observables are among others, total energy distribution (sum of all cell energy deposits), cell energy deposits distribution, longitudinal profile which represents the energy deposited by a shower as a function of the depth of the calorimeter and lateral profile which represents the energy density distribution as function of the radius of the calorimeter. Moreover, the physics-based validation process includes accuracy verification of those key distributions' first moments and precise evaluation of the tails of distributions that usually require larger amounts of samples. The GEANT4 and 3DGAN distributions are compared during this evaluation process. At inference time, a secondary validation is performed by the GEANT4 application to ensure that the fast simulation is accurate after mapping the inferred energies to positions in the calorimeter. The simulation time and the memory footprint of the model are also considered.

Once the model is trained, tested, and validated, it is deployed into the broader GEANT4 framework, and in order to do so, it is converted to a format readable by the C++ GEANT4 environment. This process is achieved using the external library for ML inference, ONNXRuntime<sup>23</sup>.

---

<sup>21</sup> ONNX: <https://onnx.ai/>

<sup>22</sup> ROOT: <https://root.cern/>

<sup>23</sup> ONNXRuntime: <https://onnxruntime.ai/>



The model architecture and the weights are also stored, respectively in the JSON HDF5 formats. This operation can be easily done in Python. Open Neural Network Exchange Runtime or ONNXRuntime is a framework for neural networks inference. After training, the model should be saved in a format that can be used for inference in C++. Then should be converted into an ONNX format using the tf2onnx<sup>24</sup> library. The disk space required for the weights, saved as HDF5 file, is about several hundreds of MBs and the model's architecture, saved as a JSON file, is hundreds of KBs.

---

<sup>24</sup> tf2onnx: <https://github.com/onnx/tensorflow-onnx>



## 3 Requirements for the thematic modules in the physics domain

**Section 3** is dedicated to reporting the requirements concerning the thematic modules to be developed for the physics domain use cases. The four physics domain use cases include lattice QCD simulations (T7.1) and particle detector simulation in High Energy Physics (T7.7), as well as noise simulation in radio astronomy (T7.2) and the VIRGO noise detector in astrophysics (T7.3).

Thematic modules requirements consolidation resulted from the activities performed so far under work package 4 (WP4), which concerns the technical co-design and validation of the digital twin engine among research communities. Under this scope the objective was to introduce use-case specific requirements for the thematic modules, based on the DTE infrastructure (WP5) and core modules (WP6). Digital twin engine core modules described in work package 6 are responsible for capabilities concerning workflow compositions, real-time acquisition of data and processing, quality and uncertainty tracing, data fusion, big data analytics, as well as AI/ML workflow.

### 3.1.1 General Description and Categorization of requirements

With respect to the DTE core modules and after research studies and analysis, institute, and community wise, the physics domain thematic modules requirements were gathered, agreeing that use cases present similar processing operations throughout their workflows. If a requirement category is not applicable to any of the use case's, this will be specifically mentioned in the respective subsection. Each requirement category is represented by a dedicated subsection following the description. The requirement categories compiling use cases' capabilities components are introduced as follows:

- **Input and output storage:** this subsection describes the input and output data requirements concerning data storage architecture utilised, HPC centres where data are available and stored, and any pre-processing methods and steps required. Moreover, the expected data volumes will be reported with respect to both input and output data.
- **Databases** subsection includes the form of databases and the database management systems that are being utilised for storage of data and metadata during use case processes.
- **Computing** is a general term that we use here to describe all the requirements related to computation resources in terms of CPUs and/or GPUs. This subsection describes the computing set up that should be provisioned for each digital twin, concerning cloud computation resources, High Performance Computing, High Throughput Computing and MPI infrastructure.

- **OS and execution framework** includes the DT requirements for the operating system and the OS-level virtualization framework for delivering DT software.
- **Machine Learning** subsection includes requirements concerning the exploitation of machine learning by each physics use case, in terms of software development language, ML frameworks, machine/deep learning models, statistical learning models, monitoring, (re)training and validation.
- **Real-time data acquisition and processing** refers to the use cases' capability of data/metadata being processed in real-time. This subsection also includes the platforms/frameworks being used for that purpose and how real-time processing is approached by the concerned use cases.
- **Data formats** subsection describes the different formats that data and metadata coming from the physics use cases present. As well as, to which formats are the original data being converted to in case of pre-processing and post-processing.
- The subsection of **software stack** describes all software tools and their requirements for each digital twin thematic module.
- **Visualisation** subsection includes the different forms and methods of visualising the results in terms of quality verification and validation.
- In the **data sharing** subsection, the processes of making data resources available are presented.

## 3.2 Storage I/O

### 3.2.1 Input data requirements

The majority of the physics thematic modules require file or object- based storage or they have already data stored in HPC centres. In particular, the T7.3 thematic module requires space on a POSIX filesystem. Same requirements apply for output data storage. Necessary pre and post processing techniques need to be applied for each one of the thematic modules, which are described in more detail in the respective subsections of [section 2](#).

### 3.2.2 Expected data volume

The data volumes that physics domain thematic modules are expecting range from O(10) GB to at most O(100) TB.

## 3.3 Databases

The use cases presented in this document do not require any external database to write and read data, but they do recognize the importance of having a service to store models' histories and metadata, such as a ML model registry. Similarly, for use case specific metadata.

## 3.4 Computing

Concerning computing resources, physics thematic modules require systems with multiple GPU support, HPC centres computing power with MPI infrastructure. As well as support for distributed computing, availability of GPUs for machine learning workloads, and ability to scale up or down computing resources based on demand.

## 3.5 OS and execution framework

To support all the related processes from modelling and ML training to inference and validation, Linux based operating systems, containerized environments (i.e., Docker, Singularity) are required at the current stage of the thematic modules.

## 3.6 Machine Learning

Most of the physics use cases use Python to perform ML related processes. Therefore, Python ML frameworks, like Tensorflow and PyTorch are being utilised for the development of the individual, use case specific machine learning frameworks. Tensorflow is the one that is used by the majority of the thematic modules. Statistics and neural network-based ML models are used, as well as monitoring tools, such as Tensorboard. Moreover, ML validation frameworks will be implemented for model validation and quality check.

## 3.7 Real-time data acquisition and processing

Most of the thematic modules in physics do not incorporate any real-time data acquisition procedure and therefore no related tools are required. Though, capabilities of online data processing are developed by T7.1 and T7.3, and especially T7.3 processes require streaming platforms, e.g., Apache Kafka. T7.2 ultimately aims at the development of a (quasi) real-time data classification tool, but this goal is for the future and is not a part of the current task, which only relies on pre-acquired data.

## 3.8 Data formats

Different data formats are produced together with different pre-processing and post-processing requirements. Therefore, data and metadata formats concerning T7.7 are ROOT and HDF5, binary and textual data for T7.1, time series data in binary form with text header for T7.2 and gwf files for T7.3.

## 3.9 Software stack

Requirements in terms of software stacks include:

- C and C++
- Python
- GEANT4
- Tensorflow, PyTorch
- JupyterLab
- Conda
- Docker, Singularity
- Workflow tools: Kubeflow, Jupyter Notebooks
- Streaming platforms: Apache Kafka
- Big data analytics tools: Apache Spark
- ML monitoring tools: Tensorboard, Prometheus, Grafana

## 3.10 Visualisation

The applications' results are going to be presented using open-source visualisation libraries and dashboards, where plots analysing the produced outcome will be displayed.

## 3.11 Data Sharing

Support for metadata management tools for tracking data lineage and ensuring data quality. Ability to share data across different teams and organisations. Tools such as GitHub and Zenodo are examples of the above.

## 4 Testbed infrastructure

In this section we report about details of testbeds used by the tasks for their PoC or early testing.

### 4.1 Fast particle detector simulation with GAN testbed

The fast particle detector simulation with GAN thematic module consists of two inseparable components as we have already discussed in [section 2.4](#). These two components are the machine learning framework and the particle simulation framework. For the ML framework the 3DGAN model ([R2](#)), has been implemented using Tensorflow. The code is available on GitHub<sup>25</sup> and it has been tested and ran on a single Linux node using 4 GPUs.

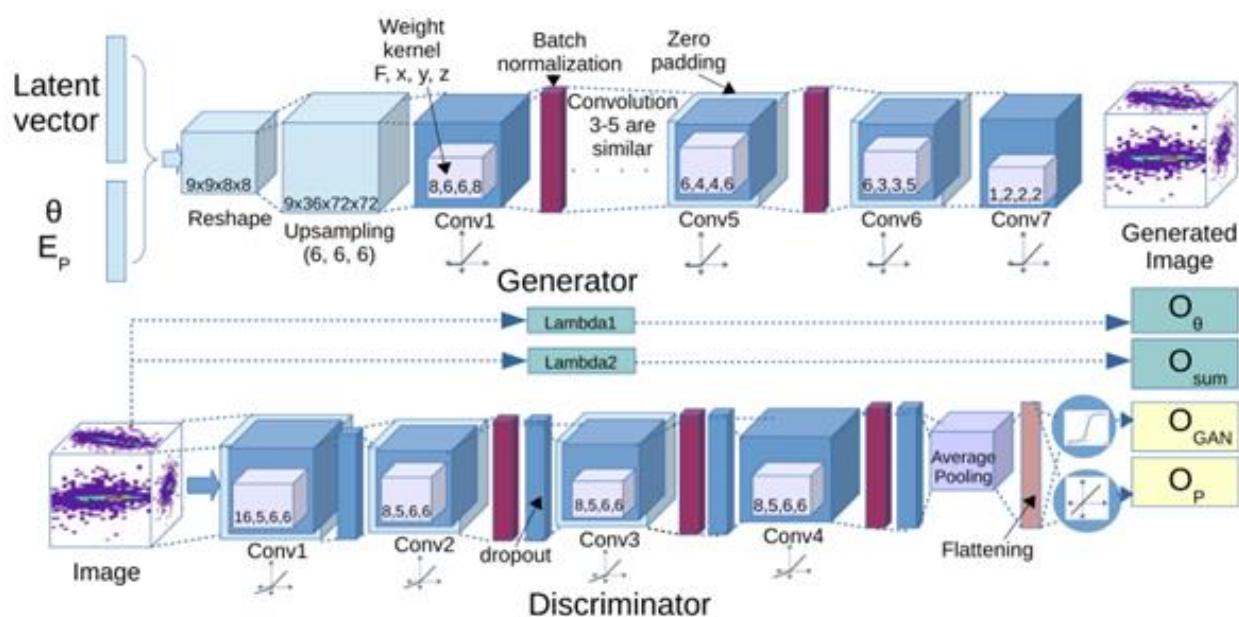


Figure 20 - 3DGAN model architecture

Concerning the particle simulation framework component of our thematic module, there have been testbeds developed that are incorporating different ML models than the 3DGAN. Therefore, our current and future efforts focus on integrating the 3DGAN model in the simulation framework that uses the GEANT4 environment and is implemented in C++. An example of the use of ML techniques for the fast detector simulation and how to incorporate inference libraries into C++ framework is the Par04 example developed by

<sup>25</sup> <https://github.com/CERN-IT-INNOVATION/3DGAN/tree/main/Accelerated3DGAN/src/Accelerated3DGAN>

the GEANT4 community and can be found on CERN Gitlab <sup>26</sup>. This example depends on the external libraries used for the ML inference, LWTNN<sup>27</sup> and ONNXRuntime. The ML model used in this example is a Variational Autoencoder (VAE), trained externally in Python on full GEANT4 detector simulation response data.

---

<sup>26</sup> <https://gitlab.cern.ch/geant4/geant4/-/tree/master/examples/extended/parameterisations/Par04>

<sup>27</sup> Lightweight Trained Neural Network: <https://github.com/lwtnn/lwtnn>



## 5 Conclusions

The Digital Twin Engine's physics thematic modules concerning tasks T7.1, T7.2, T7.3 and T7.7 have been designed and analysed throughout the first 8 months of the interTwin project. Additionally, the scientists involved in the analysis activities of WP7, and physics domain tasks have identified the technical requirements that are important for the development of the specific applications. Those requirements need to be implemented in the DTE in order for the thematic modules to be run seamlessly by community users.

In this document each physics related application has been defined and analysed in detail. More specifically, applications concern lattice QCD simulations and data management, noise simulation for radio astronomy, GAN-based thematic modules that manage noise simulation, low-latency de-noising and veto generation for gravitational waves, as well as fast particle detector simulation with GAN.

The definition of the applications and the consolidation of their requirements described above resulted in the following conclusions. All of the thematic modules presented here are interesting for the communities use cases that are leveraging state of the art machine learning advancements, and each one presents unique significance in the field's research development. Commonalities between the applications can be observed with respect to the machine learning components incorporated, as well as to their processing workflow composition. At the same time, there are particularities that describe the physics thematic modules in terms of data format produced and used, along with the relevant data processing methods utilised by each application. Finally, we observe that the nature of the majority of the use cases require modern frameworks, such as Python and ML based, to be combined with frameworks that require low level programming languages such as C/C++.



## 6 References

Reference	
No	Description / Link
<b>R1</b>	<b>Generative Adversarial Networks.</b> Ian J. Goodfellow et al. <a href="https://arxiv.org/abs/1406.2661">https://arxiv.org/abs/1406.2661</a> DOI: <a href="https://doi.org/10.48550/arXiv.1406.2661">https://doi.org/10.48550/arXiv.1406.2661</a>
<b>R2</b>	<b>Fast Simulation of a High Granularity Calorimeter by Generative Adversarial Networks.</b> Gul Rukh Khattak et al. <a href="https://arxiv.org/abs/2109.07388">https://arxiv.org/abs/2109.07388</a> DOI: <a href="https://doi.org/10.48550/arXiv.2109.07388">https://doi.org/10.48550/arXiv.2109.07388</a>
<b>R3</b>	<a href="https://g4fastsim.web.cern.ch/">https://g4fastsim.web.cern.ch/</a>
<b>R4</b>	<b>Fast Simulation for ATLAS: Atlfast-II and ISF.</b> W. Lukas. Technical Report ATL-SOFT-PROC-2012-065, CERN, Geneva, Jun (2012) DOI: <a href="https://doi.org/10.1088/1742-6596/396/2/022031">https://doi.org/10.1088/1742-6596/396/2/022031</a>
<b>R5</b>	<b>Fast simulation of the CMS detector.</b> D. Orbaker. J. Phys. Conf. Ser., (219):32–53, 2010. Part of Proceedings, 17th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2009): Prague, Czech Republic (2009). DOI: <a href="https://doi.org/10.1088/1742-6596/219/3/032053">https://doi.org/10.1088/1742-6596/219/3/032053</a>
<b>R6</b>	<b>Fast simulation of electromagnetic showers in the ATLAS calorimeter: Frozen showers.</b> E. Barberio et al. J. Phys. Conf. Ser., 160, 012082, (2009). DOI: <a href="https://doi.org/10.1088/1742-6596/160/1/012082">https://doi.org/10.1088/1742-6596/160/1/012082</a>
<b>R7</b>	<b>Auto-Encoding Variational Bayes.</b> D. P. Kingma et al. DOI: <a href="https://doi.org/10.48550/arXiv.1312.6114">https://doi.org/10.48550/arXiv.1312.6114</a>
<b>R8</b>	<b>Pixel Recurrent Neural Networks.</b> Aaron van den Oord et al. DOI: <a href="https://doi.org/10.48550/arXiv.1601.06759">https://doi.org/10.48550/arXiv.1601.06759</a>
<b>R9</b>	<b>GEANT4:</b> <a href="https://geant4.web.cern.ch/">https://geant4.web.cern.ch/</a>
<b>R10</b>	<b>HEP Software Foundation Community White Paper Working Group - Detector Simulation.</b> HEP Software Foundation. DOI: <a href="https://doi.org/10.48550/arXiv.1803.04165">https://doi.org/10.48550/arXiv.1803.04165</a>
<b>R11</b>	<b>openQ*D code: a versatile tool for QCD+QED simulations.</b> I. Campos, P. Fritzsche, M. Hansen, M. K. Marinkovic, A. Patella, A. Ramos & N. Tantalò



	<p>The European Physical Journal C, 80, Article number: 195 (2020)</p> <p>DOI: <a href="https://doi.org/10.1140/epjc/s10052-020-7617-3">https://doi.org/10.1140/epjc/s10052-020-7617-3</a></p> <p>volum</p>
<b>R12</b>	<p><b>Normalizing Flows: An Introduction and Review of Current Methods.</b></p> <p>Ivan Kobyzev; Simon J.D. Prince; Marcus A. Brubaker. IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 43, Issue: 11, 01 November 2021) DOI: <a href="https://doi.org/10.1109/TPAMI.2020.2992934">https://doi.org/10.1109/TPAMI.2020.2992934</a></p>
<b>R13</b>	<p><b>Flow-based generative models for Markov chain Monte Carlo in lattice field theory.</b> M. S. Albergo, G. Kanwar, and P. E. Shanahan Phys. Rev. D 100, 034515 – Published 22 August 2011</p> <p><a href="https://link.aps.org/doi/10.1103/PhysRevD.100.034515">https://link.aps.org/doi/10.1103/PhysRevD.100.034515</a></p>
<b>R14</b>	<p><b>Efficient modeling of trivializing maps for lattice <math>\Phi^4</math> theory using normalizing flows: A first look at scalability.</b></p> <p>Luigi Del Debbio, Joe Marsh Rossney, and Michael Wilson</p> <p>Phys. Rev. D 104, 094507 – Published 15 November 2021</p> <p><a href="https://link.aps.org/doi/10.1103/PhysRevD.104.094507">https://link.aps.org/doi/10.1103/PhysRevD.104.094507</a></p>
<b>R15</b>	<p><b>Introduction to Normalizing Flows for Lattice Field Theory</b></p> <p>Michael S. Albergo, Denis Boyda, Daniel C. Hackett, Gurtej Kanwar, Kyle Cranmer, Sébastien Racanière, Danilo Jimenez Rezende, Phiala E. Shanahan</p> <p><a href="https://doi.org/10.48550/arXiv.2101.0817">https://doi.org/10.48550/arXiv.2101.0817</a></p>