A framework for data simulation and analysis of the BabyCal electromagnetic calorimete

Hebel Lobos, Daniel León

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

"A FRAMEWORK FOR DATA SIMULATION AND ANALYSIS OF THE BABYCAL ELECTROMAGNETIC CALORIMETER"

DANIEL LEÓN HEBEL LOBOS

THESIS TO APPLY FOR THE PROFESSIONAL TITLE OF
INGENIERO CIVIL EN INFORMÁTICA

Advisor: Raquel Pezoa, Ph.D.
Co-referent: Claudio Torres, Ph.D.

July - 2023

DEDICATION

Dedicated to my beloved grandmother, *“Gueli”* Osiris Sucarrat Z. and to the memories of my beloved grandmother, *“Oma”* Rosa Weschler S. and my beloved grandfather, *“Tata”* Humberto Lobos B., for without their unconditional love and care, I would have most definitely not become the person I am today.

“No dejes apagar el entusiasmo, virtud tan valiosa como necesaria; trabaja, aspira, tiende siempre hacia la altura.”

- Rubén Darío

ACKNOWLEDGEMENTS

Thanks to my mother and father, Andrea and Arie, for always encouraging me to do better and to pursuit my own path to happiness. Thank you for putting your trust in me and for always being there when I needed you the most. Thank you for always taking care of me, supporting me along the way and for always being proud of my achievements. To my amazing family, my grandmothers, my cousins, aunts and uncles. Thank you for being always loving, supportive, and imparting me valuable wisdom and life lessons.

Thanks to my home-town friends, Raúl, Paula, Leopoldo, Marcela and Joel. For always believing that I can achieve anything I put my mind to; and thanks to my amazing girlfriend, that came into my life at the most unexpected and perfect time, for she provided me with incredible emotional support, understanding and love.

To my amazing *Squad* of friends, who accompanied me along the way during university: Macarena *Shiny* Andrade, Javiera *Su Santidad* Arce, Pedro *La Abuela* Arriagada, Braulio *Pomposo* Fuentes, Cristóbal *Peaches* Missana, John *100* Rodríguez, Bastián *Blanquito* Quezada, Fabián *Cuatro* Salas, Matías *Duende Melodía* Valenzuela and Farid *El Primo* Zalaquett, thank you for all the amazing memories. No one nor anything will ever be able to erase them. Love you all. I hope I can keep seeing all of you around.

To all the people that I met at the JIM (*Jornada de Inserción Mechona*) organization, thank you, It was a true honor being part of such an amazing team. One of the highlights of university was bumping into you people every day. To a group of friends that included me during one of the most stressful times I've ever had during my studies: Vicho, Axel, Tralma, Claudio, Pedro, Nacha, Diego, Yisus, Caco, Joaquin and Chon, thank you for all your support, I will be forever grateful. To my friends, Mono, Dieguito, Dani, Oby, Rojito, Biki, Rhodo, Pola, Fanny, Seba, Cotorra, Yess, Cata, Kein, Stephy and Quesito: Thank you from the heart, you are all an important part of my life at university. Know that you can always count on me.

Thanks to my supervising teacher, Raquel Pezoa, for guiding me along the amazing journey that was this thesis work. Thank you for always being there when I needed help, knowledge and inspiration to persevere and be consistent. Thanks to my co-guiding teacher, Claudio Torres, for being part of this project, and for teaching me so much throughout my time at university since the first semester.

To my teachers Cecilia Reyes, Hubert Hoffmann and Luis Hevia: the amount work you put into the informatics department to be the great entity it is today, is something only worthy to be thankful for and most certainly to look up to. For working every day to try make this university a better place for everyone. For that and so much more, thank you.

Huge thanks to Maurizio Ungaro at Jefferson Lab in Virginia, USA, Esteban Molina and Bruno Benkel at CCTVal in Valparaíso for supporting this project and supporting me as well by helping along the way. This work was partially supported by FONDECYT Postdoc project #3190740 and ANID PIA/APOYO AFB180002.

ABSTRACT

***Abstract*—** This thesis work presents an automated system capable of efficiently simulating, translating, and analyzing high-energy physics (HEP) data generated by the simulated BabyCal electromagnetic calorimeter. By leveraging HEP data simulation software, computer clusters, and cutting-edge machine learning algorithms, such as convolutional neural networks (CNNs) and autoencoders, the system effectively manages a dataset of approximately 10.000 entries. Using the framework, we generated simulated data of muon and antimuon particles and implemented CNNs and autoencoders to analyze it. The framework was tested and evaluated with muon and antimuon particles. This led to an interesting challenge, from the computational point of view, regarding the differentiation between the two. The analysis showed that muons and antimuons exhibit a lot of behaviour similarities when colliding with the simulated BabyCal with the used spatial representation. The experiment results showed that autoencoders were able to reconstruct muons, achieving accuracies of up to 98%. This work is a starting point that serves as a helpful data analysis tool, aiding researchers in their investigations.

***Keywords*—** Computer Clusters; High Energy Physics; Data Simulation; Data Analysis; Machine Learning

RESUMEN

***Resumen*—** Este trabajo de título presenta un sistema automatizado capaz de simular, traducir y analizar eficientemente datos de física de alta energía (HEP) generados por el calorímetro electromagnético simulado *BabyCal*. Mediante el uso de software de simulación de datos HEP, clústeres de computadoras y algoritmos de aprendizaje automático, incluidas redes neuronales convolucionales (CNN) y *autoencoders*, el sistema logra procesar con éxito un *dataset* de aproximadamente 10.000 entradas. Usando el *Framework*, se generaron datos simulados de partículas de muones y antimuones, además de implementarse CNNs y *autoencoders* para analizar los datos. El *Framework* se probó y evaluó con partículas de muones y antimuones. Esto condujo a un desafío interesante, desde el punto de vista computacional, con respecto a la diferenciación entre los dos. El análisis mostró que los muones y los antimuones exhiben muchas similitudes de comportamiento cuando chocan con el BabyCal simulado usando en el caso de la representacioón utilizada. Los resultados del experimento mostraron que los *autoencoders* pudieron reconstruir muones, logrando precisiones de hasta un 98%. Este trabajo es un punto de partida que sirve como una herramienta útil de análisis de datos, ayudando a los investigadores en su labor.

***Palabras Clave*—** Clústeres de computadores; Física de Alta Energía; Simulación de datos; Análisis de datos; Aprendizaje automático

GLOSSARY

AI: Artificial Intelligence

ATLAS: The Atlas Experiment

BCAL: Barrel Calorimeter

CCTVal: Centro Científico Tecnológico de Valparaíso

CERN: European Council for Nuclear Research

CLAS: CEBAF Large Acceptance Spectrometer

CNN: Convolutional Neural Network

Docker: Containerization Framework

EVIO: Event IO

Geant 4: Geant4 Simulation Toolkit

GEMC: GEant 4 Monte Carlo

GPU: Graphics processing unit

HEP: High Energy Physics

HPC: High-Performance Cluster

IBM: International Business Machines Corporation

LHC: The Large Hadron Collider

ML: Machine Learning

Singularity: Containerization Framework

UTFSM: Universidad Técnica Federico Santa María

Contents

List of Figures

List of Tables

INTRODUCTION

High energy physics researchers are constantly seeking innovative tools to efficiently handle, analyze, and understand vast amounts of data [CCTVal, 2021]. This thesis work introduces an automated system designed to simulate, translate, and analyze HEP data generated by a simulated electromagnetic calorimeter named BabyCal. By combining HEP data simulation software, computer clusters, and machine learning algorithms like convolutional neural networks (CNNs) and autoencoders, this system manages to bring synergy, functionality and scalability to a system that did not have that before, turning it to a framework.

This work focuses on leveraging the capabilities of the developed automated system, which effectively manages a dataset of more than 10,000 entries. By utilizing HEP data simulation software and harnessing the computational power of computer clusters, the system demonstrates high efficiency in processing considerable amounts of data, a crucial aspect for HEP data analysis.

In this work, the framework is tested and evaluated with muon and antimuon particles. This led to an interesting challenge, from the computational point of view, regarding the differentiation between the two. The analysis showed that muons and antimuons exhibit a lot of behaviour similarities when colliding with the simulated BabyCal with the used spatial representation.

To dive deeper into this observation, an autoencoder is trained exclusively on muon data and used to reconstruct them. The system achieves significant accuracy, reaching up to 98%. This finding suggests that, within the given dataset and data representation that was used, muons and antimuons lacked distinct spatial features, posing challenges and significant difficulties for their classification.

Despite this challenge, the development of the automated system holds significant implications for the rapidly advancing world of HEP data analysis. Researchers can now employ this tool to expedite their investigations and gain new insights. The system's capabilities enable efficient data simulation, translation, and analysis with state-of-the-art machine learning algorithms. Moreover, this thesis work highlights the need for larger and more diverse datasets to gain a deeper understanding of particle classification in HEP.

CHAPTER 1
PROBLEM DEFINITION

1.1 WHAT IS A FRAMEWORK?

In programming, a software framework is usually a tool that provides a standard way to build and deploy applications. They usually include support programs, compilers, code libraries, toolsets, and application programming interfaces (APIs).

In this context, that is not the definition of Framework being referred to. Instead, this is the one to be used throughout this thesis work:

"A framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful."
[CODATA, 2020]

That being said, the purpose of the Framework being created in this thesis work, is to integrate and automate a process that is currently not fully functioning as a system; because its parts do not really communicate with each other and don't have the required synergy.

1.2 THE STANDARD MODEL OF PARTICLE PHYSICS

The theories and discoveries of thousands of physicists since the 1930s have resulted in a remarkable insight into the fundamental structure of matter:

All matter is made of elementary particles, specifically quarks and leptons, which are arranged in generations based on their stability and mass. There are four fundamental forces in the universe: strong, weak, electromagnetic, and gravitational. Three of these forces are carried by bosons, or force-carrier particles, while the graviton is believed to carry gravity, though it has not yet been found. Our best understanding of how these particles and three of the forces are related to each other is encapsulated in the Standard Model of particle physics.

The Standard Model is the current best description of the subatomic world, but it only includes three of the four fundamental forces and does not explain dark matter or the antimatter mystery. Developed in the early 1970s, it has successfully explained almost all experimental results and precisely predicted a wide variety of phenomena.

Over time and through many experiments, the Standard Model has become established as a well-tested physics theory. The Higgs boson was discovered in 2012 [Aad et al., 2012], [Chatrchyan et al., 2012], and although it is an essential part of the Standard Model, the theory is still incomplete. The hope is that new experiments at the Large Hadron Collider will reveal more about the subatomic world and the missing pieces of the puzzle [CERN, 2022].

All particles can be summarized as follows (Figure 1):

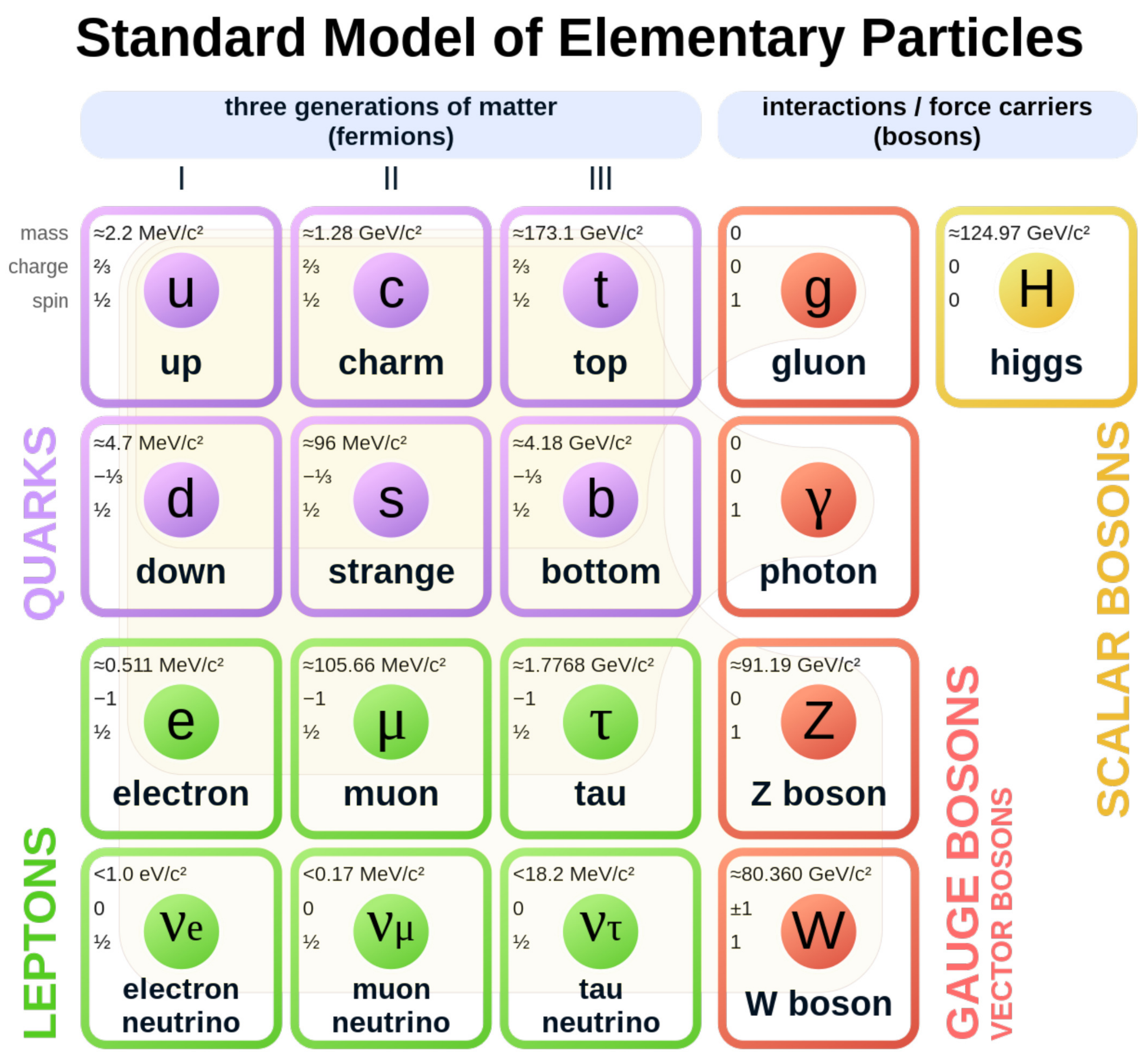


Figure 1: Diagram of The Standard Model of Elementary Particles, `Source:` `Wikipedia`

1.3 BACKGROUND

The following section and subsections are, in its majority, directly extracted text out of the scientific paper "Construction and performance of the barrel electromagnetic calorimeter for the GlueX experiment" [Beattie et al., 2018], which describes in detail the further mentioned investigation. The purpose of this section is to tell a bit of the story that led to the current use of the BabyCal Electromagnetic Calorimeter at Universidad Técnica Federico Santa María.

The barrel calorimeter is part of the new spectrometer installed in Hall D at Jefferson Lab for the GlueX experiment. The calorimeter was installed in 2013, commissioned in 2014 and has been operating routinely since early 2015. The detector configuration is described herein. The calorimeter records the time and energy deposited by charged and neutral particles created by a multi-GeV photon beam. It is constructed as a lead and scintillating-fiber calorimeter and read out with 3840 large-area silicon photomultiplier arrays. Particles impinge on the detector over a wide range of angles, from normal incidence at 90 degrees down to 11.5 degrees, which defines a geometry that is fairly unique among calorimeters [Beattie et al., 2018].

1.3.1 GLUEX DETECTOR

The primary motivation of the GlueX experiment is to search for and, ultimately, study the pattern of gluonic excitations in the meson spectra produced in γp collisions at 9 GeV.

Gamma-p (γp) collisions are a type of scattering process that occur in particle physics, where a high-energy photon (γ) interacts with a proton (p) target. In these collisions, the photon transfers some of its energy and momentum to the proton, causing it to scatter at a particular angle, which can be detected and measured.

Gamma-p collisions are of particular interest to particle physicists because they can provide information about the structure of the proton at very small scales. By measuring the scattering angles and energies of the particles produced in these collisions, physicists can probe the internal structure of the proton and study the distribution of its constituent quarks and gluons.

One important aspect of gamma-p collisions is that the photon has zero charge and is not affected by the strong nuclear force, which makes it an ideal probe for studying the internal structure of the proton without being influenced by its strong interactions. In addition, the high energy of the photon allows for very precise measurements of the proton's structure, allowing physicists to study the proton in greater detail than was previously possible.

Gamma-p collisions are typically carried out at high-energy particle accelerators such as the HERA (Hadron Elektron Ring Anlage) accelerator at DESY in Germany, where high-energy electrons are used to produce high-energy photons through the process of synchrotron radiation. These photons are then collided with protons in the HERA storage ring, allowing physicists to study the structure of the proton in great detail.

Specifically, GlueX aims to study the properties of hybrid mesons — particles where the gluonic field contributes directly to the J^{PC} [1] quantum numbers of the mesons. The design of the GlueX detector is based on a solenoidal magnet that surrounds all detectors in the central region, providing a magnetic field of about $2T$ along the direction of the photon beam, which impinges on a 30 cm-long liquid hydrogen target. A schematic of the detector including its major sub-detectors is given in Figure 2. The goal of GlueX calorimetry is to detect and to measure photons from the decays of π^0's and η's and other radiative decays of secondary hadrons. The detector measures the energies and positions of the showers made by photons, as well as the timing of the hits. It also provides the timing of the hits caused by charged hadrons, allowing for time-of-flight particle identification [Beattie et al., 2018].

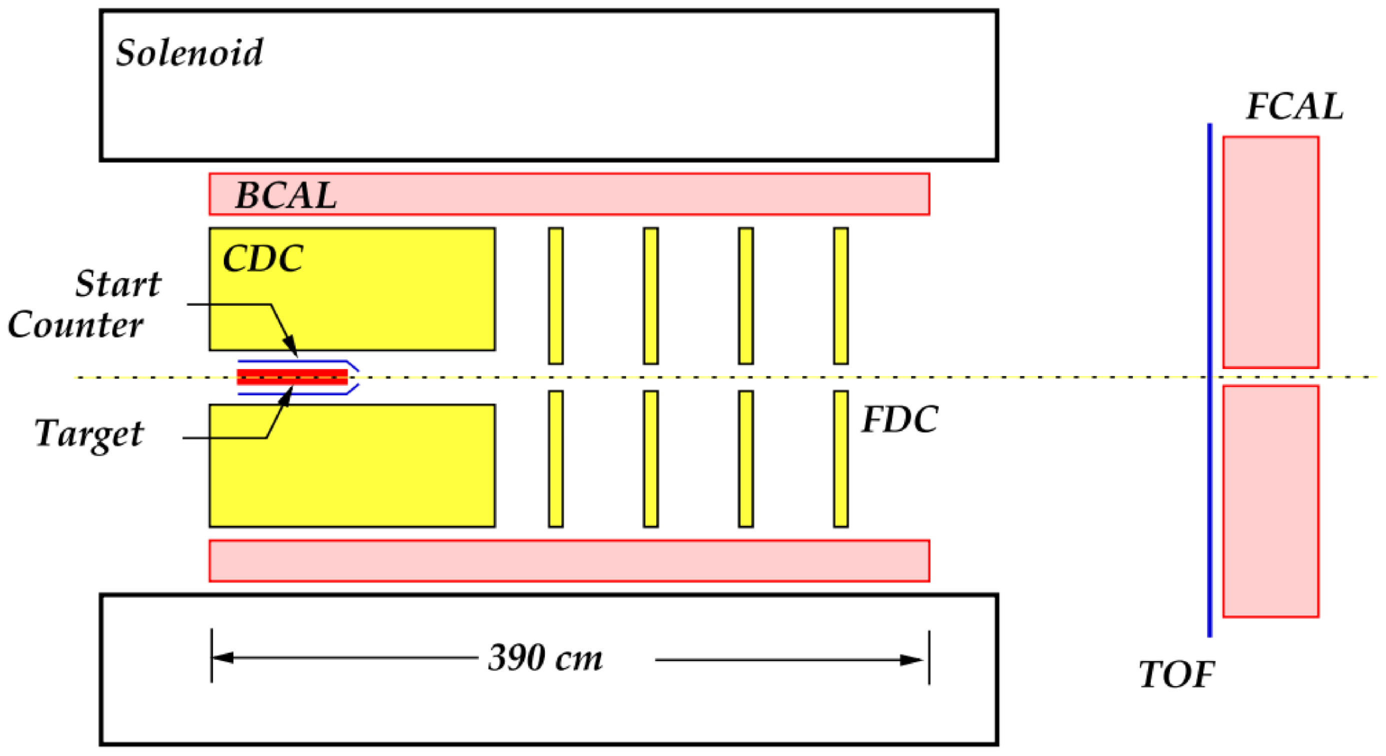


Figure 2: Sketch of GlueX detector. The main systems of the detector are the Start Counter, the Central Drift Chamber (CDC) the Forward Drift Chamber (FDC), a scintillator-based Time of Flight (TOF) wall and a lead-glass Forward Calorimeter (FCAL). The Barrel Calorimeter (BCAL) is sandwiched between the drift chambers and the inner radius of the solenoid [Beattie et al., 2018]. Source: [Beattie et al., 2018]

[1] J^{PC}: Quantum numbers of elementary particles used for their classification.

1.3.2 BCAL: OVERVIEW AND DESIGN

A practical solution to the requirements and constraints imposed by the experiment is a calorimeter based on lead-scintillating fiber sandwich technology. The BCAL is modeled closely after the electromagnetic calorimeter built for the KLOE experiment at DAΦNE. The BCAL detects photon showers with energies between 0.05 GeV and several GeV, $11^{\circ} - 126^{\circ}$ in polar angle, and $0^{\circ} - 360^{\circ}$ in azimuthal angle. Geometrically, the BCAL consists of 48 optically isolated modules each with a trapezoidal cross section, forming a 390 cm-long cylindrical shell having inner and outer radii of 65 cm and 90 cm, respectively. The fibers run parallel to the cylindrical axis of the detector. Schematics showing the geometry of the BCAL and readout segmentation are shown in Figure 3 [Beattie et al., 2018].

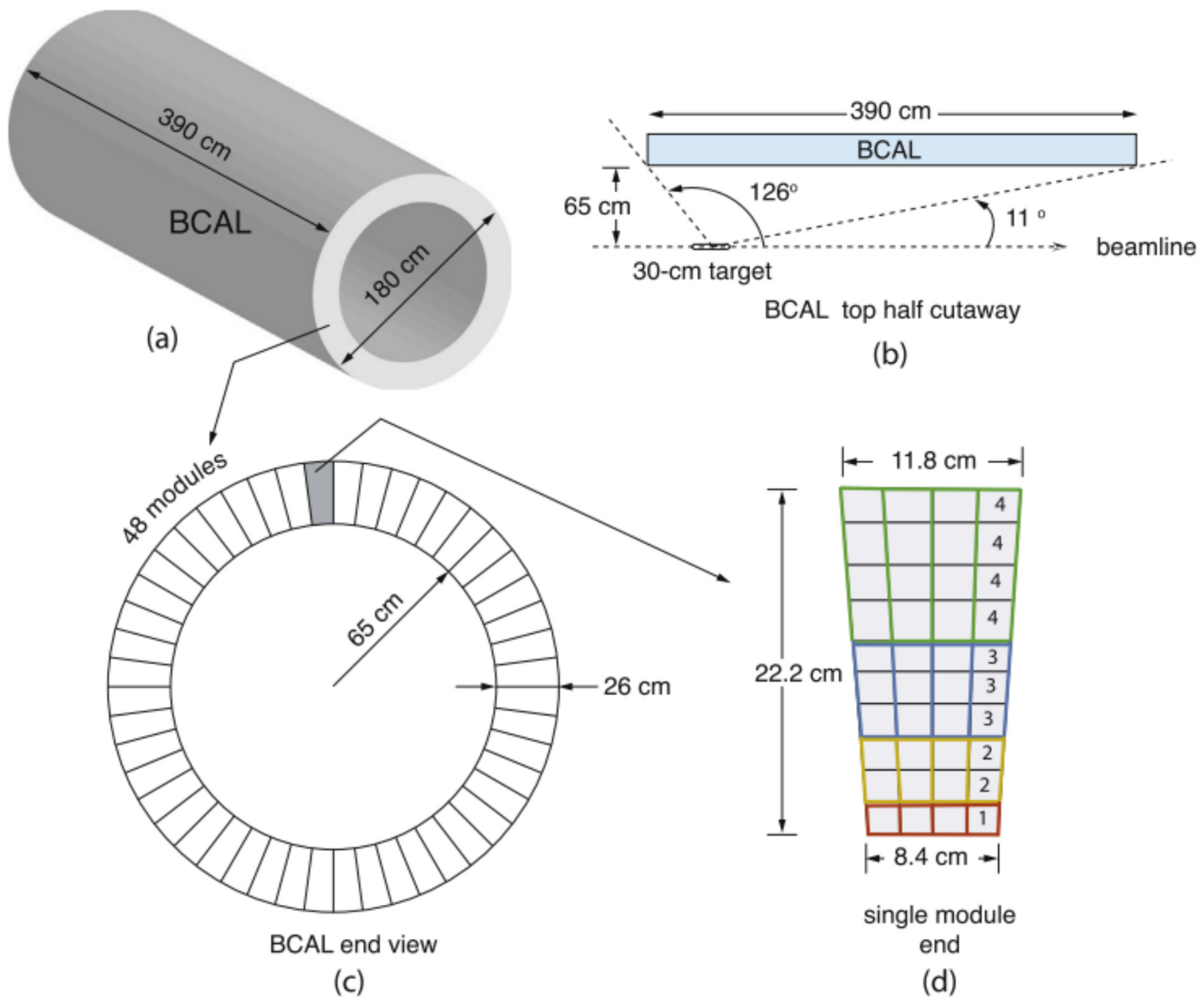


Figure 3: Sketch of the Barrel Calorimeter and readout. (a) A three-dimensional rendering of the BCAL; (b) top-half cutaway (partial side view) of a BCAL module showing its polar angle coverage and location with respect to the GlueX LH2 target; (c) end view of the BCAL depicting all 48 azimuthal modules and (d) wedge-shaped end view of a single module showing the location of light guides and sensors [Beattie et al., 2018]. Source: [Beattie et al., 2018]

Finally, as a way of picturing the BCAL the best way possible and to better understand the way the BabyCal works, part of the assembly process of the whole detector, specifically the BCAL, is shown in Figure 4. This is relevant and it relates to the BabyCal Electromagnetic Calorimeter because the BabyCal is the prototype used for the BCAL in the Glue-X experiment and is now located at the Physics Department of Universidad Técnica Federico Santa María. The following section, will emphasize on the problem at hand regarding the BabyCal and the Framework that needs to be designed and implemented.

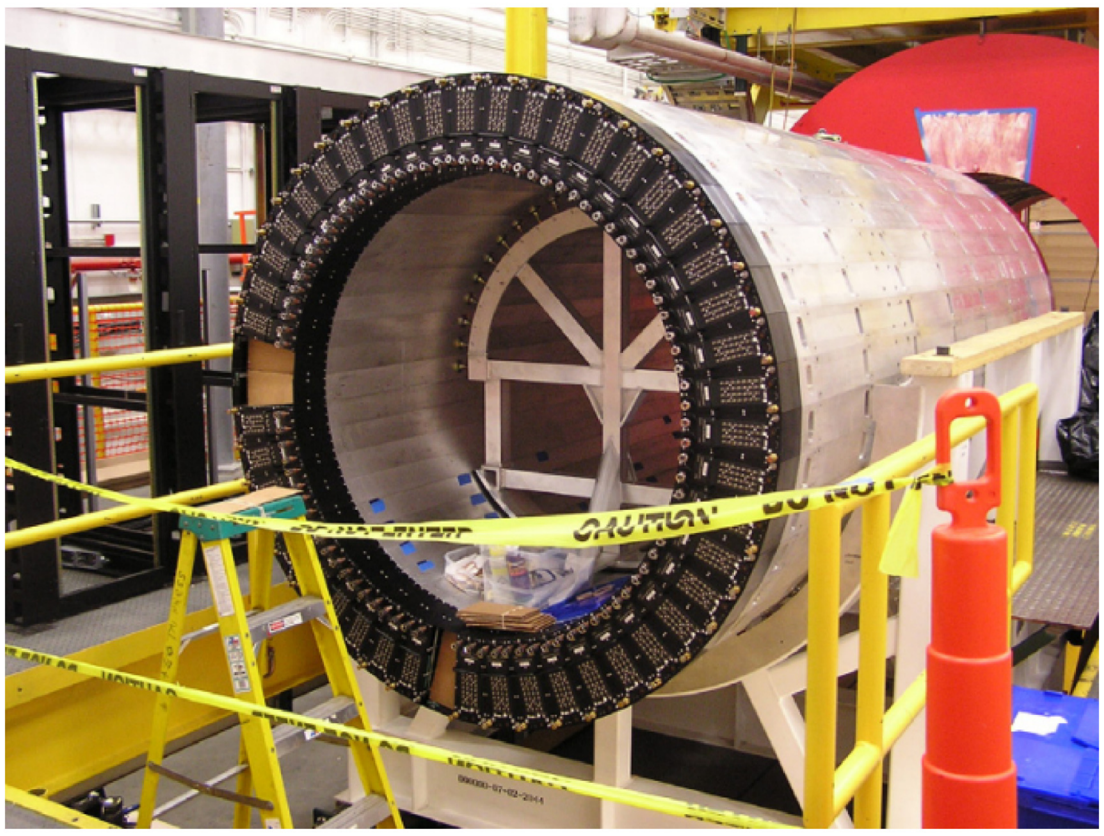


Figure 4: BCAL assembly before insertion into the bore of the magnet (red yoke in the top right corner). Most of the electronic packages (black) have been mounted on the end of each module but cables are not yet connected. Also visible inside the BCAL is a temporary fixture to support the upper modules during assembly [Beattie et al., 2018]. Source: [Beattie et al., 2018]

1.4 CONTEXT

The Physics Department at Universidad Técnica Federico Santa María (UTFSM) currently has a sampling electromagnetic calorimeter called ***BabyCal*** that was used as a prototype for the Barrel Calorimeter of the Glue-X experiment at Jefferson Lab [Beattie et al., 2018] in Virginia, USA. Currently, the device has the potential to be used for detecting multiple types of particles. The BabyCal Calorimeter is composed by a two-dimensional array of optical fibers that are placed parallel on top of each other (like a matrix), with a specific geometrical configuration (spacing) between them. At the same time, the fibers having a long, cylindrical shape, generate a cubical structure for the calorimenter. Figure 5 shows a front view of the device, showing the illuminated scintillating optical fibers and Figure 6 shows the complete look and structure of the BabyCal.

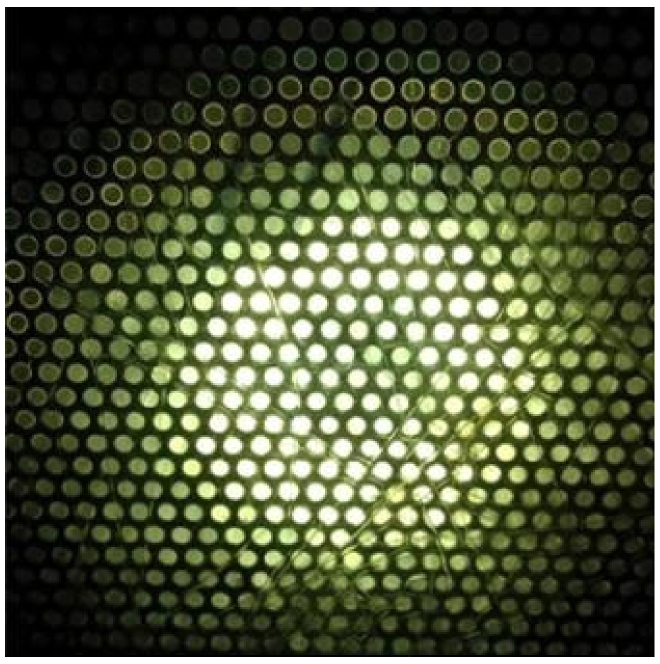


Figure 5: Front view of the BabyCal Electromagnetic Calorimeter. `Source: CCTVal UTFSM`

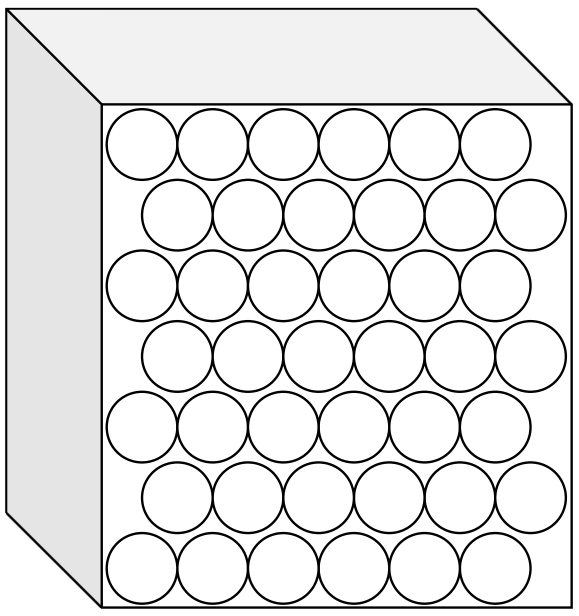


Figure 6: Schematic representation of the BabyCal Electromagnetic Calorimeter: The circles represent the faces of the optical fibers. They are stacked together parallel to each other. `Source: Own elaboration`

Even though the device could get prepared to be used in the generation of experimentally tested HEP[2] data at UTFSM, the challenge is to make use of machine learning-based algorithms to automatically analyze said data and isolate the information of interest from the background signal. This cannot be achieved by simply executing one experiment after another manually. Mainly, because there's the need of very high volumes of HEP data to train whatever machine learning tool is needed to complete such a task. Instead, a software tool called GEMC[3] will be used to simulate the BabyCal's interaction with the different types of particles.

The computational resources required for processing and storing the expected high volume of HEP data are hosted by the CCTVal[4] Data Center, which has been continuously providing support for data analysis for over a decade. It is an operative node (a Tier-2) of the LHC[5] grid [CERN, 2023a], providing computing services to the ATLAS[6] experiment at CERN[7]. The CCTVal collaborated with the Jefferson Lab in Virginia at various experiments [CCTVal, 2021] like the CLAS12[8] experiment [JLAB, 2021] and the GlueX experiment [Beattie et al., 2018]. The CCTVal Data Center currently has about 800 CPU cores of processing capability and a storage capacity of 300 TB. They also provide Graphics Processing Units (GPU) support with more than 10 GPU NVIDIA Graphics cards.

1.4.1 GEMC

GEMC allows the user to simulate the BabyCal in any environment by choosing certain parameters that will guide the simulation and generate the needed data. To be more precise, GEMC is also a framework in itself, because it is a real software structure that is built on top of the CERN's GEant4 Simulation Toolkit [CERN, 2023b] and it uses it to simulate the passage of particles through matter (see Figure 7). Thus, giving the user a new useful tool with which to analyze the data coming from a detector, in this case, the BabyCal Electromagnetic Calorimeter [Ungaro, 2019].

GEMC provides:

- Application independent geometry description
- Easy interface to build / run experiments
- CAD/GDML imports (geometry formats) [CCIN2P3, 2012]

[2]High Energy Physics
[3]**GE**ant4 **M**onte-**C**arlo
[4]Centro Científico Tecnológico de Valparaíso
[5]The Large Hadron Collider
[6]ATLAS Experiment
[7]European Council for Nuclear Research
[8]**C**EBAF **L**arge **A**cceptance **S**pectrometer

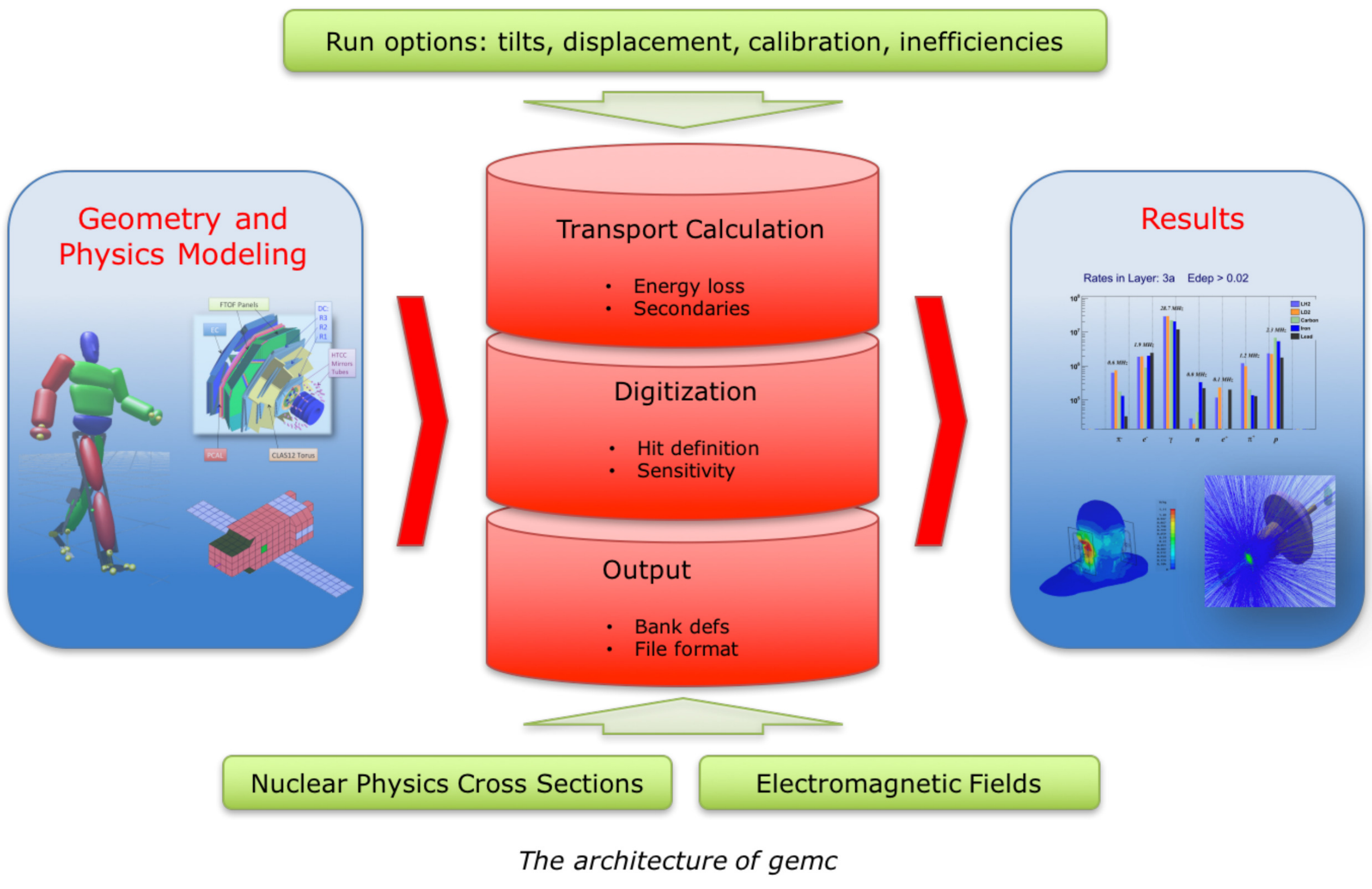


Figure 7: GEMC Architecture, Source: GEant4 Monte-Carlo

The advantage about GEMC, is that due to its modular usability, the simulation process can be automated in order to generate the amount of data needed to train a machine learning tool. Nevertheless, it is not possible to directly pass the simulated data to said tool, because most modern machine learning tools use a typically standard format determined by the programming language and the analysis toolkit intended for the process. In this case, the usual target format are Python's *"numpy"*[9] matrices. Therefore, the data needs to be translated first. For this obstacle to be overcome, a tool called *Gruid Translator* will be used. It is important to mention that *"translating"* in the context of this thesis work, means transforming the output format of the simulated data to another that allows for easier handling for the data analysis.

[9]Numpy Python Library

1.4.2 GRUID TRANSLATOR

The Gruid Translator is a small tool in Python that attempts to convert a GEMC simulation of a generic calorimeter detector to a standard output format. Taking a text file (`.txt`) exported by the GEMC simulation, this tool generates a time series of sparse matrices detailing the location and energy deposited of hits in the scintillating fibers [Benkel, 2021]. As this is a modular tool, it can, like GEMC's simulation process, be automated as well. Later in this work in Chapter 3 (The Framework), some units will be used to describe part of the Gruid Translator's I/O mechanism (from now on referred as Gruid for short). They are the following:

- Energy is measured in **MeV**.
- Distance is measured in **cm**.
- Time is measured in **ns**.

1.5 ABOUT THE CURRENT DATA HANDLING PROCESS

The current HEP data treatment process goes like the following:

1. **Generate simulated HEP data with GEMC's simulation functionality:** A configuration file is filled with the desired parameters for the simulation and proceeds to be executed on the CCTVal's Cluster or in a local PC. GEMC runs the file and generates the simulated HEP data like it would be if the BabyCal was used to detect particles.
2. **Translate the simulated HEP data with the Gruid Translator to a "*Pythonic*" format (Sparse matrix represented in a JSON format):** The resulting HEP data of the simulation is passed on to said tool, with which it gets translated to a much more comprehensible format. This allows an easier analysis and understanding of the data being handled.
3. **Analyze the HEP data with traditional Python programming:** Analyze the data to understand its behavior and apply the desired metrics with user-available tools.

1.6 IDENTIFIED PROBLEMS

The following problems are the ones identified in the before-mentioned data handling process.

- **Data generation and extraction**
  - The need to generate many simulations.
  - Not having expert knowledge on the handling of GEMC.
  - The need to generate a large volume of data from said simulations to train a machine learning tool.
- **Data transformation**
  - The need to format data from many simulations to deliver to a machine learning tool in order to train it.
- **Data analysis**
  - The need to analyze data from many simulations using a machine learning tool.
- **There is no automated integration of the simulations with the data analysis of the experiments that want to be carried out with the BabyCal.**
- **There is no Framework for data simulation and analysis.**
  - Impossibility to generate many simulations in an automated way.
  - Insufficient HEP data volume to train a machine learning tool.
  - Uselessness of a machine learning tool for HEP data analysis.

The identified problems can be structured and classified into a problem tree for an easier understanding, making it simpler to address each one of them. The relevant classifications will be:

- Causes (problem starters)
- The main problem
- Effects (consequences)

Figure 8 shows the identified problems structured into the problem tree.

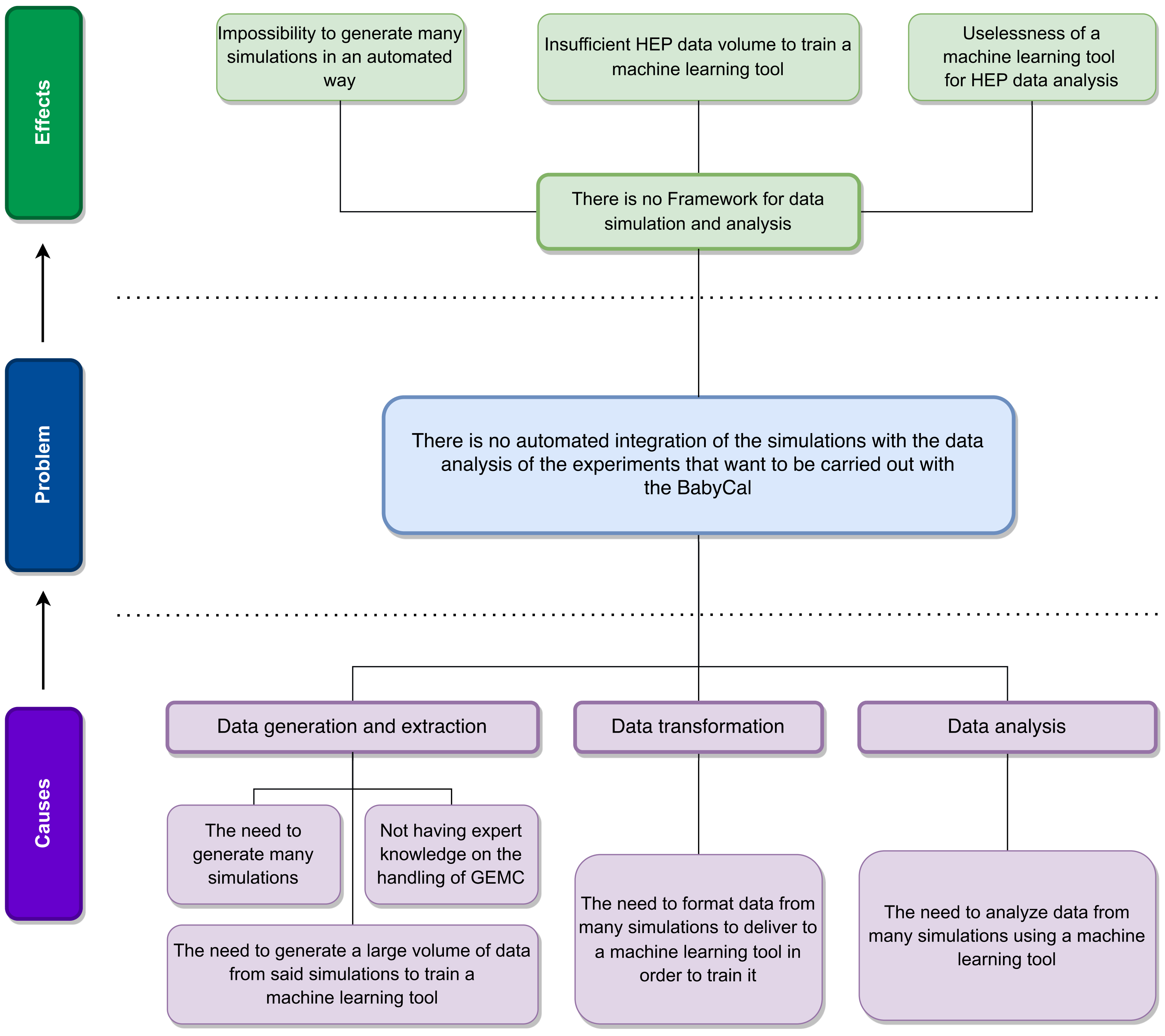

Effects
Impossibility to generate many simulations in an automated way
Insufficient HEP data volume to train a machine learning tool
Uselessness of a machine learning tool for HEP data analysis
There is no Framework for data simulation and analysis
Problem
There is no automated integration of the simulations with the data analysis of the experiments that want to be carried out with the BabyCal
Causes
Data generation and extraction
Data transformation
Data analysis
The need to generate many simulations
Not having expert knowledge on the handling of GEMC
The need to generate a large volume of data from said simulations to train a machine learning tool
The need to format data from many simulations to deliver to a machine learning tool in order to train it
The need to analyze data from many simulations using a machine learning tool

Figure 8: Problem Tree, `Source: Own elaboration`

1.7 OBJECTIVES

1.7.1 MAIN OBJECTIVE

The main objective of this thesis will be to design and implement a framework for integrating the simulation and analysis of HEP particle data of the BabyCal Electromagnetic Calorimeter in computer clusters.

1.7.2 SPECIFIC OBJECTIVES

- Automate data simulation using GEMC on a computational cluster.
- Transform the simulated data using the Gruid Translator.
- Analyze the simulated particle physics data using a machine learning-based method.

1.8 CONTRIBUTION

The main contribution of this work is the development of a framework to automate and integrate the simulation and analysis of HEP data for the BabyCal electromagnetic calorimeter. This work will be focused on the creation of said framework and the analysis of BabyCal events to successfully create a machine learning tool that allows for such a task. The machine learning approach will be used to design, implement, and validate the proposed method. The validation is a crucial step in this work, because the proposed algorithm will be tested on the BabyCal framework also to be created.

CHAPTER 2
CONCEPTUAL FRAMEWORK

2.1 AUTOMATION

The IBM site on automation refers to the concept as the use of technology to perform tasks with where human input is minimized. This includes enterprise applications such as business process automation (BPA), IT automation, network automation, automating integration between systems, industrial automation such as robotics, and consumer applications such as home automation and more [IBM, 2023].

It involves the design, development, and deployment of systems that can operate and execute functions automatically, following predefined rules or instructions. It often involves as well, the integration of hardware, software, and control systems to create automated workflows and processes. The benefits of automation include increased productivity, improved accuracy, reduced operational costs, enhanced safety, and faster task completion.

2.2 COMPUTER CLUSTERS

Computer clusters are interconnected groups of computers or servers that work together as a unified system. They are designed to enhance performance by distributing computational tasks among multiple machines. Clusters are typically used for high-performance computing (HPC) applications that require significant processing power, such as scientific simulations, data analysis, and large-scale computing.

One of the main advantages of using clusters is their ability to handle large workloads by dividing them into smaller tasks that can be executed in parallel across multiple nodes. This parallel processing capability enables faster execution and improved performance on the task being handled.

Additionally, clusters provide high availability and fault tolerance as they can continue functioning even if some nodes fail, thanks to its redundant architecture. For this specific thesis work, a high-performance cluster (HPC) is going to be used (CCTVal's HPC Cluster). HPCs are specifically designed for demanding computational tasks and rely on a lot of hardware, such as powerful processors and large memory capacities.

2.2.1 JOBS - SERIAL VS PARALLEL EXECUTION

Executing jobs in series means each task is performed one after another, sequentially. However, utilizing the cluster more efficiently involves running jobs in parallel rather than in series. By doing so, multiple tasks can be executed simultaneously, allowing for faster task speed and faster overall results [NM State University, 2021].

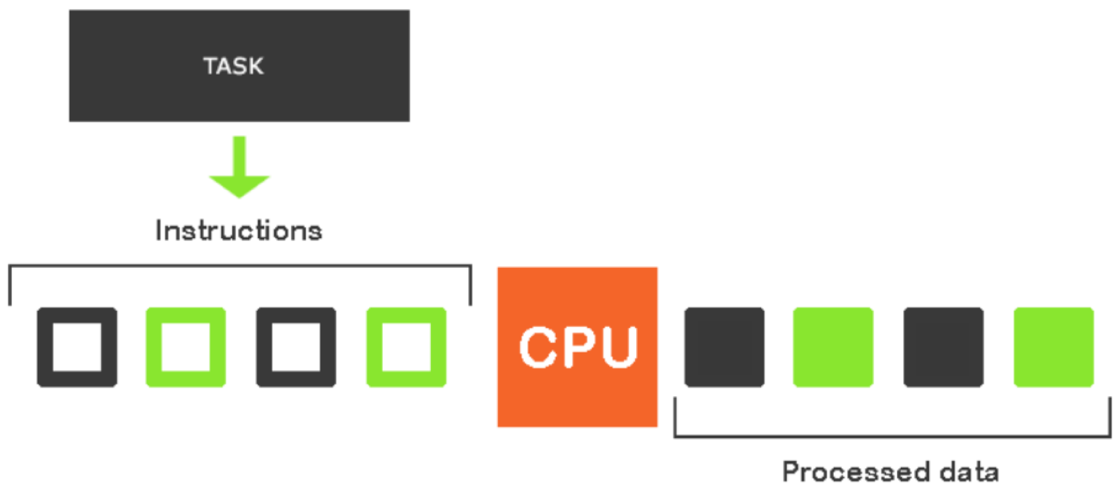


Figure 9: Serial Job Execution, `Source:` `[NM State University, 2021]`

Figure 9 illustrates the sequential execution of a task by a single processor, which is suitable for basic applications with minimal computational requirements. However, when dealing with complex and time-intensive programs that demand significant processing power and speed, it becomes necessary to explore parallel computing. By transitioning to parallel processing, it is possible to accelerate the task completion and achieve faster results [NM State University, 2021].

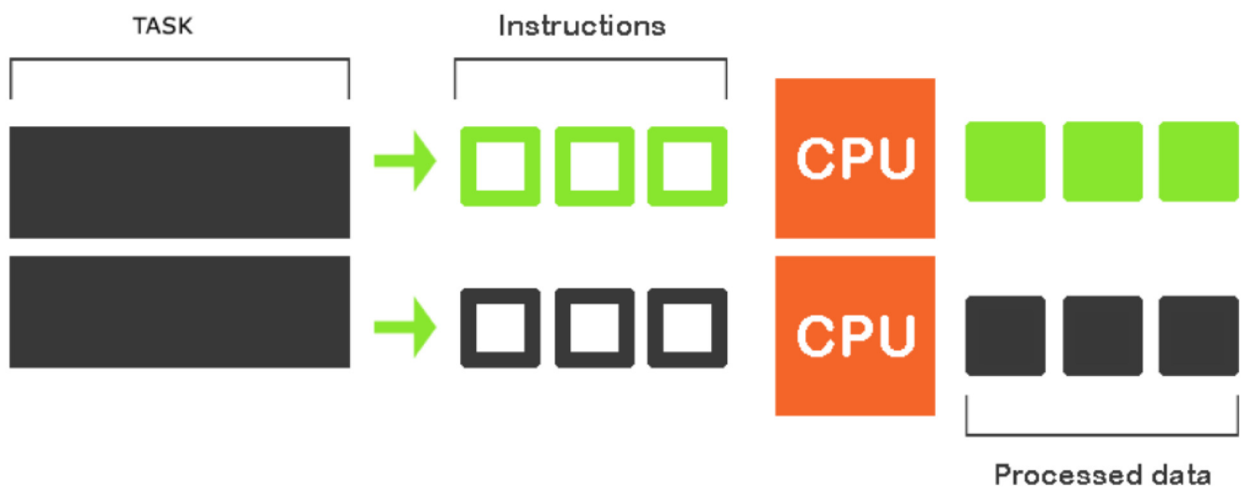


Figure 10: Parallel Job Execution, `Source:` `[NM State University, 2021]`

Figure 10 illustrates the concurrent execution of multiple tasks by multiple CPUs. By increasing the number of CPUs, the tasks can be completed more swiftly, as the workload is distributed among the processors, leading to enhanced efficiency and faster results [NM State University, 2021].

2.3 SINGULARITY CONTAINERS

Singularity containers are a type of lightweight, portable, and reproducible software packaging technology used in high-performance computing (HPC) and scientific computing environments. Singularity enables the creation and deployment of application containers that encapsulate an entire software stack, including the operating system, libraries, and dependencies, in a single, self-contained unit.

Unlike traditional containerization with Docker, Singularity containers are designed to prioritize compatibility with scientific and HPC workloads. Singularity containers are primarily intended for running applications in shared computing environments, such as clusters and supercomputers, where users may not have administrative privileges or the ability to modify the host system [Singularity Docs, 2019].

One of the key features of Singularity is its ability to provide a level of isolation between the host system and the containerized application while allowing the application to interact directly with the underlying system resources. This feature enables users to run applications within Singularity containers as if they were running natively on the host system, without the need for virtualization or root access [Gerber, 2018].

2.4 MACHINE LEARNING: STATE OF THE ART

Machine learning is a branch of *artificial intelligence* that involves using data and training algorithms to recognize patterns in it. Machines are able to learn how to identify common elements on specialized datasets and use this knowledge to recognize them on new ones. By analyzing patterns and information, machine learning systems enhance their understanding and decision-making abilities. This iterative process allows computers to learn and make predictions without explicit programming. Thus enabling computers to acquire knowledge and improve their performance over time. Instead of providing explicit instructions, the computer system is trained on a dataset containing examples or patterns, and it learns to identify and generalize patterns or relationships from the data. This training involves using mathematical and statistical techniques to create models that can make predictions or decisions based on new data. During the last few years, machine learning approaches have become increasingly crucial for analyzing complex HEP data [Radovic et al., 2018] [Albertsson et al., 2018] [Baldi et al., 2014] [Guest et al., 2018].

There are different types of machine learning algorithms, the three main paradigms are supervised learning, unsupervised learning, and reinforcement learning:

2.4.1 SUPERVISED LEARNING

In supervised learning, the algorithm learns from a labeled dataset, where each example is associated with a known output or target. The algorithm learns to map input data to the correct output by finding patterns or relationships in the labeled data. It can then make predictions or classifications on unseen data. A classic way of using supervised learning is in the realm of image classification, where after being trained, the user can, for example, give the model an image with various animals, and the model will be able to distinguish each one (see Figure 14). This is achieved by previously training the machine with big amounts of labeled images, so it knows what to expect later. See Figure 11.

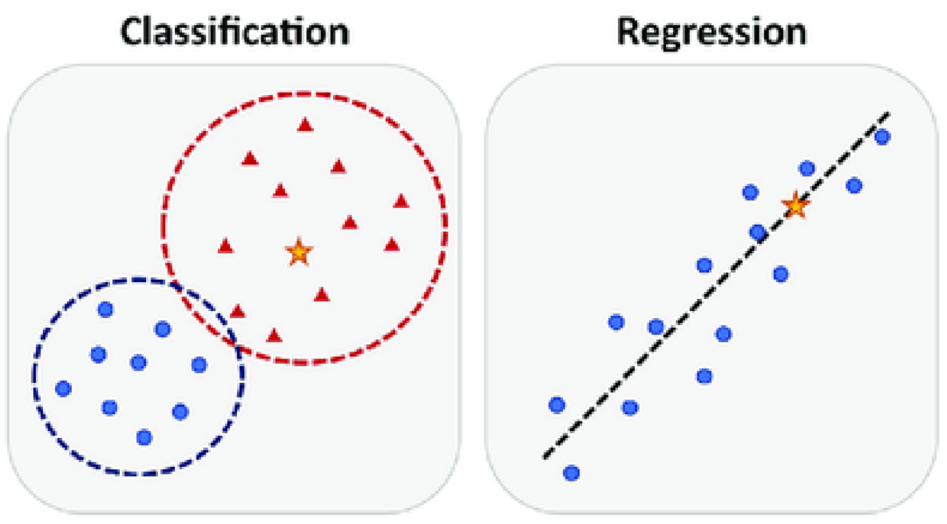


Figure 11: Supervised Learning, `Source:` [Peng et al., 2021]

2.4.2 UNSUPERVISED LEARNING

Unsupervised learning involves learning from an unlabeled dataset, where the algorithm aims to find patterns, structures, or relationships within the data without any pre-existing labels or targets. It can be used for tasks such as clustering similar data points or dimensionality reduction. See Figure 12.

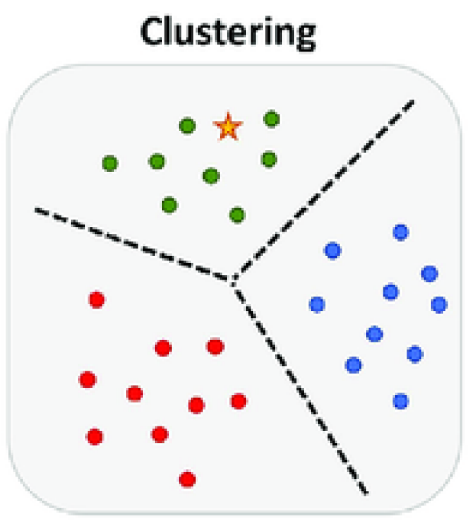
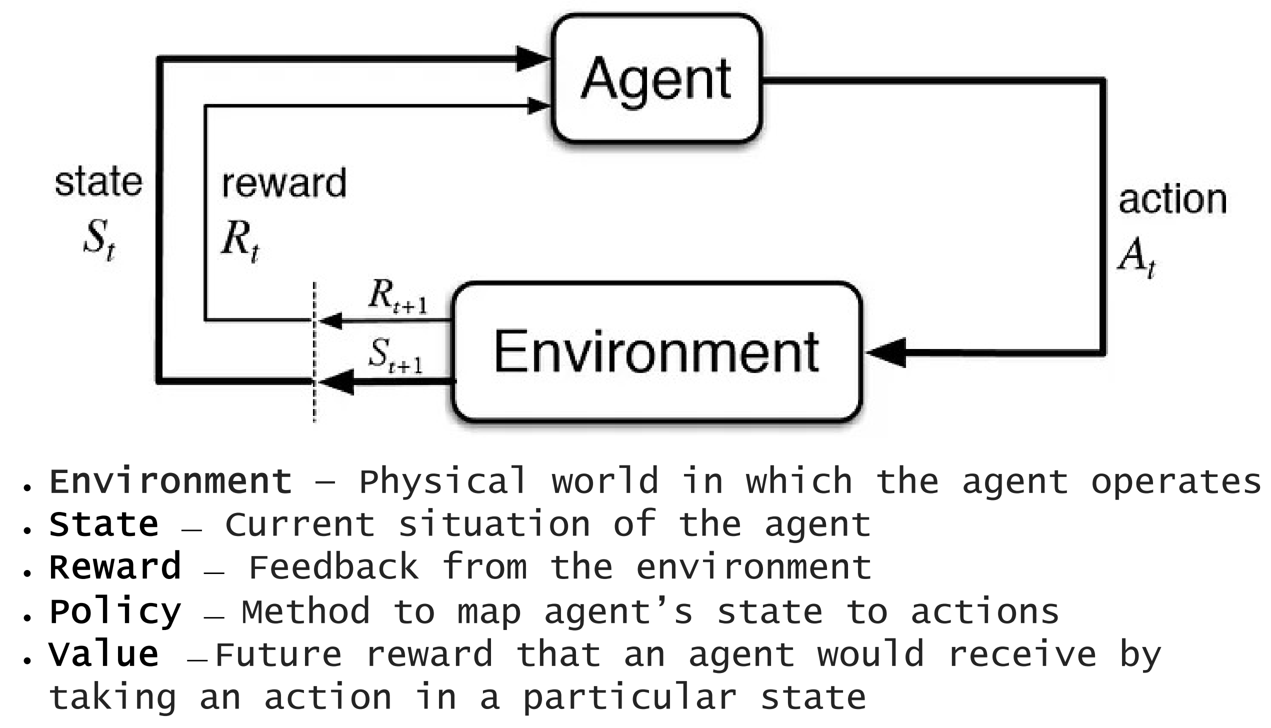


Figure 12: Unsupervised Learning, `Source:` [Peng et al., 2021]

2.4.3 REINFORCEMENT LEARNING

Reinforcement learning involves an agent learning to make decisions and take actions in an environment to maximize a reward signal. The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions. Over time, it learns the optimal strategy or policy for achieving its goal. A very good example are now-a-days' adaptive level video games. They challenge the player according to how the player is performing during playtime. See Figure 13.



- **Environment** – Physical world in which the agent operates
- **State** – Current situation of the agent
- **Reward** – Feedback from the environment
- **Policy** – Method to map agent's state to actions
- **Value** –Future reward that an agent would receive by taking an action in a particular state

Figure 13: Reinforcement Learning, `Source:` [Bhatt, 2018]

2.5 DEEP LEARNING

Deep learning is the current machine learning approach that is revolutionizing data analysis in many fields, improving the state-of-the-art performance on image and speech recognition, genomics research, natural language processing, and many other domains [Lecun et al., 2015]. Hosny et al., have described deep learning as follows: "Deep learning is a subset of machine learning that is based on a neural network structure loosely inspired by the human brain. Such structures learn discriminative features from data automatically, giving them the ability to approximate very complex nonlinear relationships" [Hosny et al., 2018].

Fully-connected networks and convolutional neural networks are two supervised deep learning techniques that have been successfully used in the analyzing of HEP data coming from particle detectors. Thus, the application of deep learning methods to the analysis of the digitized signals produced by the particles that will transverse the BabyCal has the potential to be the right tool to dominate the task at hand.

2.5.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a type of deep learning architecture that have been widely utilized in various fields, including computer vision and pattern recognition. CNNs have shown remarkable performance in analyzing and extracting meaningful features from complex data, making them particularly well-suited for image analysis tasks.

The key component of CNNs is the convolutional layer, which applies a set of learnable filters to the input data. These filters, also known as kernels, slide across the input to detect different patterns and spatial features. The convolution operation allows the network to capture local correlations and hierarchies present in the data. By stacking multiple convolutional layers, pooling layers, and fully-connected layers, followed by non-linear activation functions (commonly the Softmax function), CNNs can learn increasingly abstract representations of the input data. See figure 14.

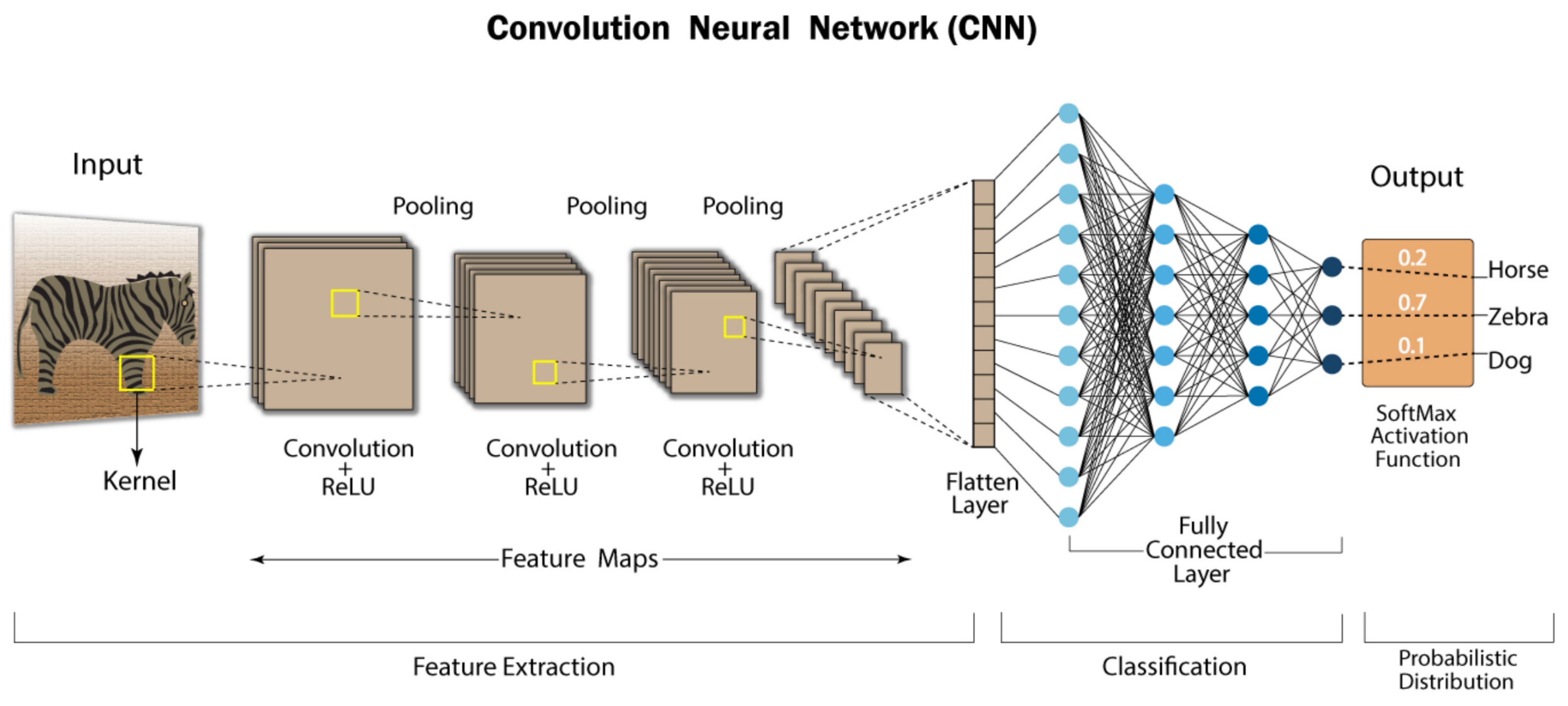


Figure 14: Convolutional Neural Network example. Source: [Swapna, 2020]

2.5.2 AUTOENCODERS

An autoencoder is a type of neural network architecture used for unsupervised learning tasks. It is designed to reconstruct the input data from a compressed representation known as the "latent space". The size of the latent space in the autoencoder can significantly impact its accuracy and performance. Its size determines how much information is retained and how effectively the original features are reconstructed. A smaller latent space might force the autoencoder to discard intricate details, leading to an oversimplified representation and lower accuracy. So t's important to not choose a size that is too small even though there's no theoretical limit for its size. In the same way, a larger latent vector can result in overfitting, where the model merely memorizes the training data without generalizing well to new data. Thus, finding the right balance in latent vector size is crucial, as it directly influences the trade-off between model complexity and accuracy.

The autoencoder consists of an encoder network, which compresses the input data into a lower-dimensional representation, and a decoder network, which reconstructs the original data from the compressed representation. See Figure 15

The autoencoder's training objective is to minimize reconstruction error, the difference between original input and reconstructed output. Accurate reconstruction means learning a compressed representation that captures essential features of the input.

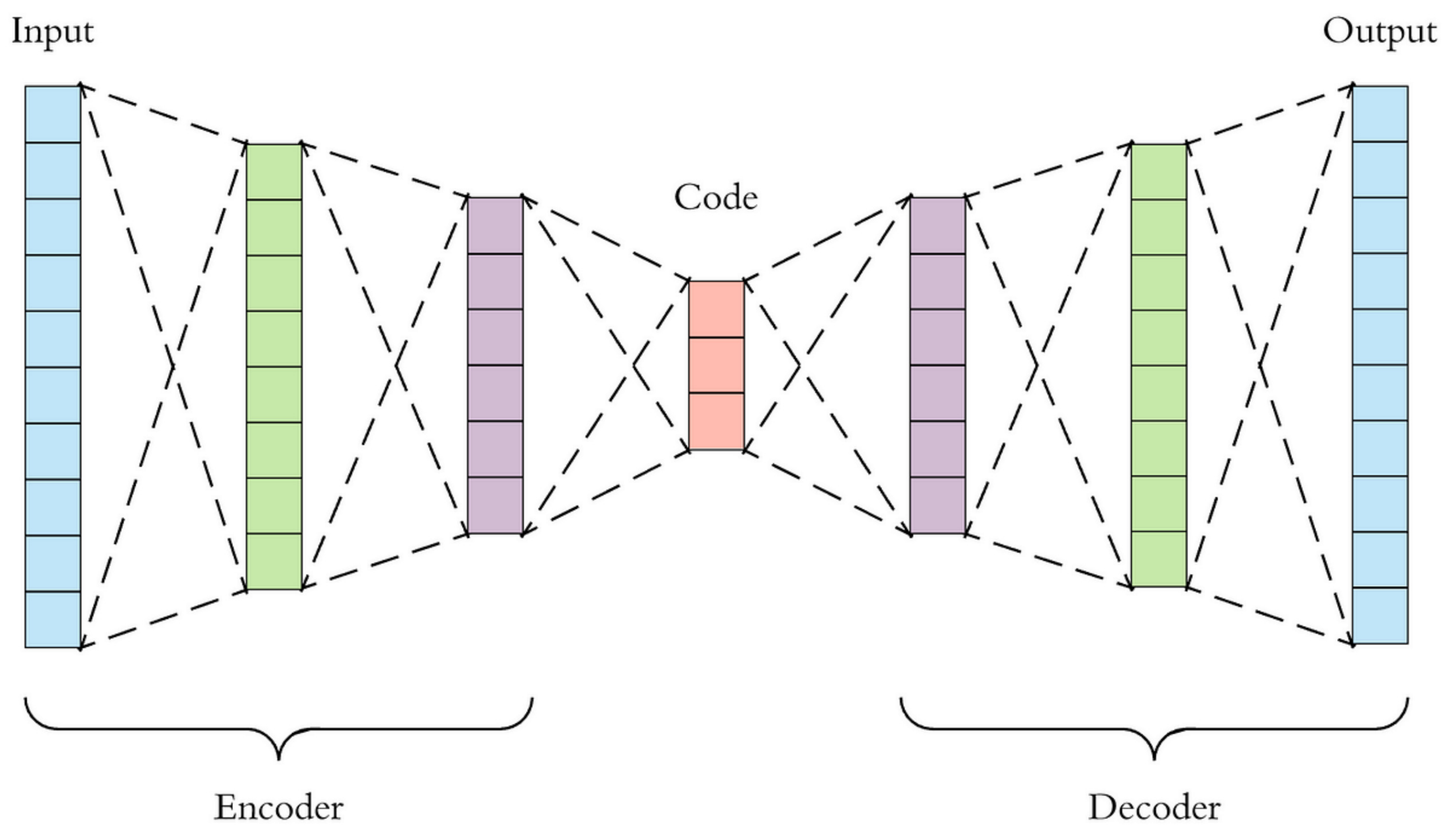


Figure 15: General autoencoder architecture; "Code" represents the latent space. Source: [Dertat, 2017]

2.5.3 ACTIVATION FUNCTIONS

In this section, a couple of activation functions relevant to this work will be mentioned. Activation functions play a crucial role in artificial neural networks, forming the backbone of deep learning models. By introducing non-linearity, these functions enable neural networks to learn complex patterns and make accurate predictions.

- **ReLU (Rectified Linear Unit)**:

  ReLU is the most popular activation function in modern deep learning models. It outputs the input if it is positive and zero otherwise, introducing simplicity and avoiding vanishing gradient problems.

$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$
$$= max(0, x) = x\mathbf{1}_{x>0}$$

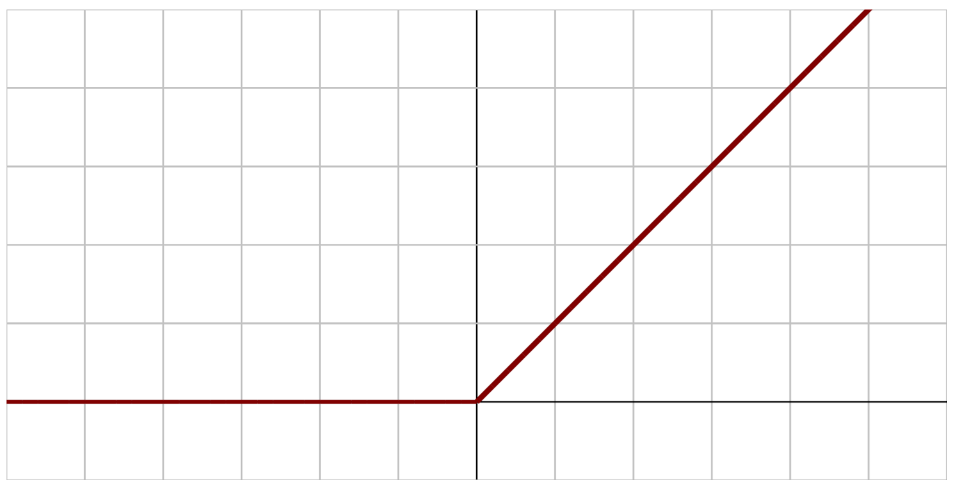


Figure 16: ReLU Activation Function, Source:Laughsinthestocks - Own work, CC BY-SA 4.0, Wikipedia Series on Machine Learning

- **Softmax**:

  Softmax is commonly used in the output layer of a classification model. It transforms the final layer's raw scores into a probability distribution, allowing the model to make class predictions.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \text{ for } i = 1, ..., J$$

2.5.4 PERFORMANCE METRICS

In this section, a couple of relevant performance metrics relevant to this work will be mentioned [Powers, 2020]. Performance metrics are essential tools for evaluating the effectiveness and efficiency of machine learning models. They measure how well a model performs its intended task. Most of them are calculated based on the Confusion Matrix (see 1, which is a performance metric that summarizes the results of the predictions in a classification task scenario.

Table 1: Confusion Matrix for binary classification instance. `Source: Own Elaboration`

	Prediction 0	**Prediction 1**
Real 0	TN	FP
Real 1	FN	TP

From here, a set of definitions ought to be considered:

- True Positive (TP) : Result is positive, and was predicted positive.
- False Negative (FN) : Result is positive, but was predicted negative.
- True Negative (TN) : Result is negative, and was predicted negative.
- False Positive (FP) : Result is negative, but was predicted positive.

To better understand the confusion matrix and the values, the following association can be applied:

- Positive or Negative: refers to the prediction. If the model predicts 1, it will be positive, and if it predicts 0, it will be negative.
- True or False: refers to whether the prediction is correct or not.

The performance metrics that were most relevant for the development of this thesis work are the following:

- **Accuracy**:

  Accuracy is one of the most fundamental metrics used to evaluate the performance of a model. It measures the amount of correct predictions made by the model compared to the total amount of predictions.

While accuracy is a valuable metric, it might not be suitable for imbalanced datasets, where one class dominates the others. In such cases, high accuracy can be misleading, and other metrics should be used to provide a more comprehensive view of the model's performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision and Recall**:

  Precision and recall are metrics commonly used in classification tasks. Precision represents the proportion of true positive predictions (correctly predicted positive instances) to the total predicted positive instances.

  $$\text{Precision} = \frac{TP}{TP + FP}$$

  Recall, on the other hand, measures the proportion of true positive predictions to the total actual positive instances. These metrics are especially important when dealing with imbalanced datasets, as they provide insights into the model's ability to identify positive instances correctly while avoiding false positives.

  $$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score**: The F1 score is the harmonic mean of precision and recall, providing a balanced evaluation metric for classification tasks. It is especially useful when precision and recall need to be considered together, as it balances their importance and helps to evaluate the overall performance of the model.

  $$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Mean Squared Error (MSE)**:

  MSE, or Mean Squared Error, is a widely used performance metric in machine learning prediction tasks. It measures the average squared difference between the predicted values and the actual values in the dataset.

  MSE quantifies the magnitude of prediction errors, and it provides a way to evaluate how well a model fits the data. Smaller values of MSE indicate better model performance, as they suggest that the model's predictions are closer to the actual values.

  One advantage of MSE is that it penalizes larger errors more significantly due to the squaring operation, making it sensitive to outliers. However, this sensitivity can also be a limitation when dealing with datasets that have extreme outliers, as it may overemphasize their impact on the overall error. This makes it convenient to use when dealing with balanced dataset that lack extreme outliers, since it provides a reliable prediction.

CHAPTER 3
THE FRAMEWORK

3.1 SYSTEM OVERVIEW

The provided diagram (Figure 17) depicts the complete framework (from now on also referred as "The Framework") that will be implemented. The following sections will provide explanations of each component, detailing their individual operations and functionalities.

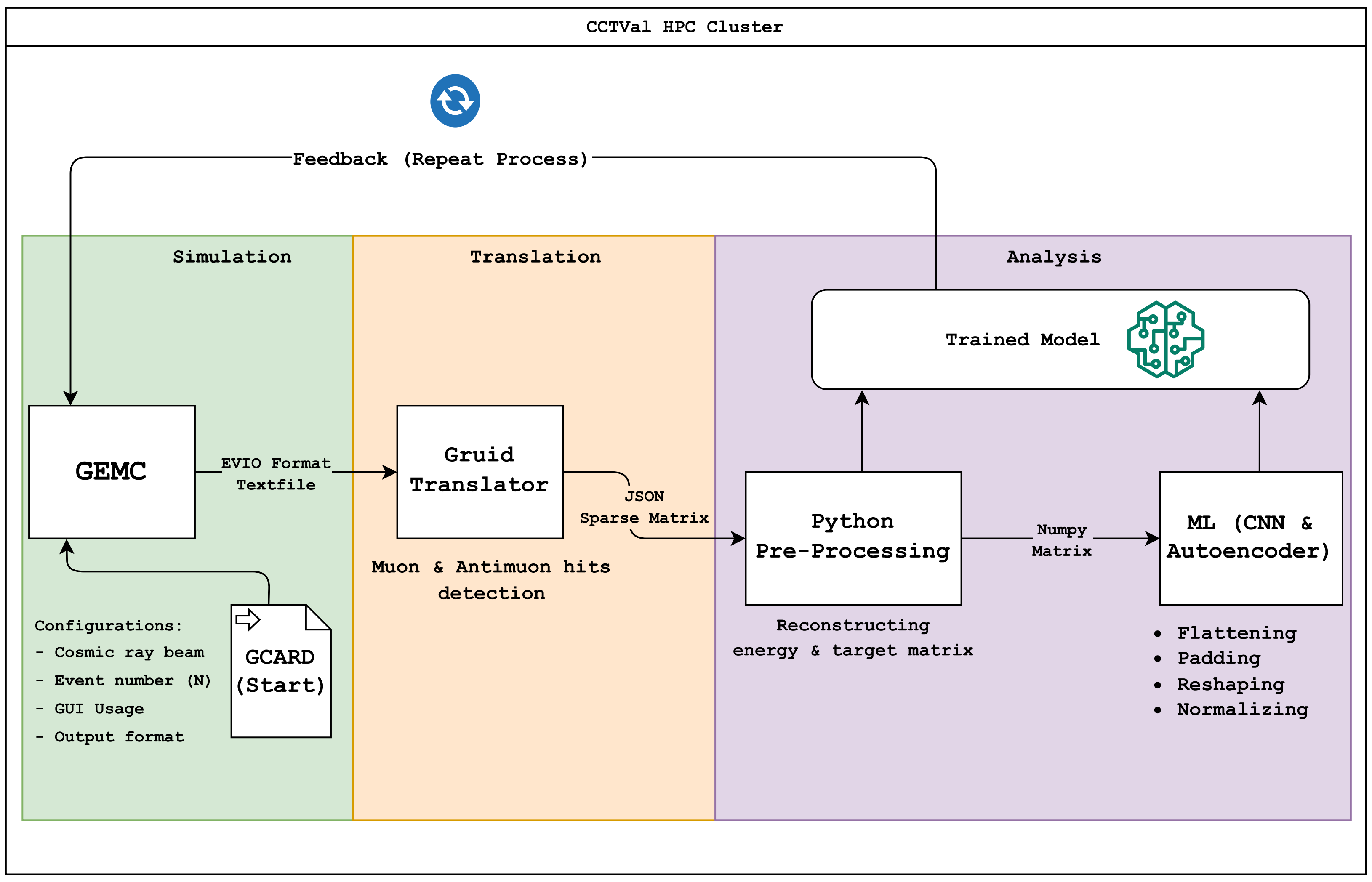


Figure 17: Simplified Framework Diagram. The image depicts the whole process of the framework, from the simulation with GEMC, through the translation with Gruid, till the Python pre-processing and the analysis with Machine Learning Techniques. `Source: Own elaboration`

3.2 SIMULATION

GEMC, being a third-party software tool and a comprehensive framework, will be incorporated into the system as a *"black box"* component. Due to its ongoing development, it is essential to select a specific version that guarantees compatibility with other system components and consistency in the output data.

During the evaluation and familiarization process with the tool, two significant versions of GEMC, namely **GEMC v2** and **GEMC v3**, were considered for potential utilization in this project.

3.2.1 GEMC v2 vs. GEMC v3

In this section, a comparison table is provided to assess the different aspects, pros, and cons of GEMC v2 and v3. This analysis aims to determine the most suitable GEMC version for constructing the framework:

Table 2: Comparison between versions of GEMC 2 & GEMC 3. `Source: Own elaboration`

Considerations	**GEMC 2**		**GEMC 3**
Geant 4 Version	10.6	4.10.06	11.0.3
Gruid Translator compatibility	Partial to none	Full	None
Cluster compatibility	Singularity version only	Native (Compilable)	Singularity version only
BabyCal Architecture compatibility	Full	Full	None
GCARD input compatibility	Full	Full	None
Output format	EVIO	EVIO	ROOT
Architecture programming language	Perl	Perl	Python 3
Installation, maintenance and deployment	Very Hard	Hard	Normal

For this thesis work, the developer's version of GEMC v2 has been chosen as the preferred option. Despite utilizing the older Geant4 v4.10.06 engine, this version offers crucial compatibility with CCTVal's Cluster and the simulated BabyCal's architecture [Molina, 2021], allowing for the complete automation of the system. Moreover, selecting GEMC v2 enables smooth integration with the other components of the framework, ensuring a cohesive workflow. The decision to utilize this version aims to maximize the benefits provided by GEMC v2 while maintaining compatibility and integration within the overall system architecture.

GEMC's structure as a software tool can be summarized in four main files:

- `bcal.pl`: It's the main script. It calls all the other scripts for the whole simulation execution.
- `bcal_geometry.pl`: Contains the geometry specifications for the different components of the detector
- `bcal_materials.pl`: Contains all the custom materials definitions.
- `bcal.gcard`: (Shown in the diagram (see Figure 17) as `GCARD`) Can be defined as the "initial input" script.

3.2.2 GEMC INPUT HANDLING

GEMC's input handling looks fairly simple in the diagram, but in reality, it has two main stages. The first one is more complex, while the second one refers to the actual input for GEMC, which is the GCARD.

1. **Compile the BabyCal architecture:** This step requires the PERL code files mentioned prior to this section that define the geometry and the materials for the detector that GEMC will simulate. These are vital for tuning the parameters of the GCARD, since the dimensions of the simulated detector are fixed for the rest of the process. If the user wishes to simulate collisions with different detector dimensions, they must rebuild the BabyCal regenerating the geometry and material files. For this thesis work, the data set generated by the simulations used a BabyCal configuration of 15 rows and 15 columns of fibers.

2. **Tune the simulation parameters with the GCARD:** This corresponds to the actual input for the simulation part of the process. Figure 18 shows one of the GCARDs used to simulate the Muon particle (μ^-) collisions for this thesis work. The language format used by the GCARD is very similar to HTML. The parameters that were tuned in these experiments were `N` (the number of events in one simulation; this is the number of cosmic rays that collide with the detector), `BEAM_P` (Beam particle, momentum, angles (in respect of the z-axis), `OUTPUT` (to automate the creation of timestamp identifiers for the generated files) and `USE_GUI` (to enable visual testing and *in background* execution for automation). The rest of the parameters were maintained fixed to ensure the physical coherence of each experiment. Further insight on GEMC's different options can be found at GEMC's Legacy Documentation Page: GEMC Options.

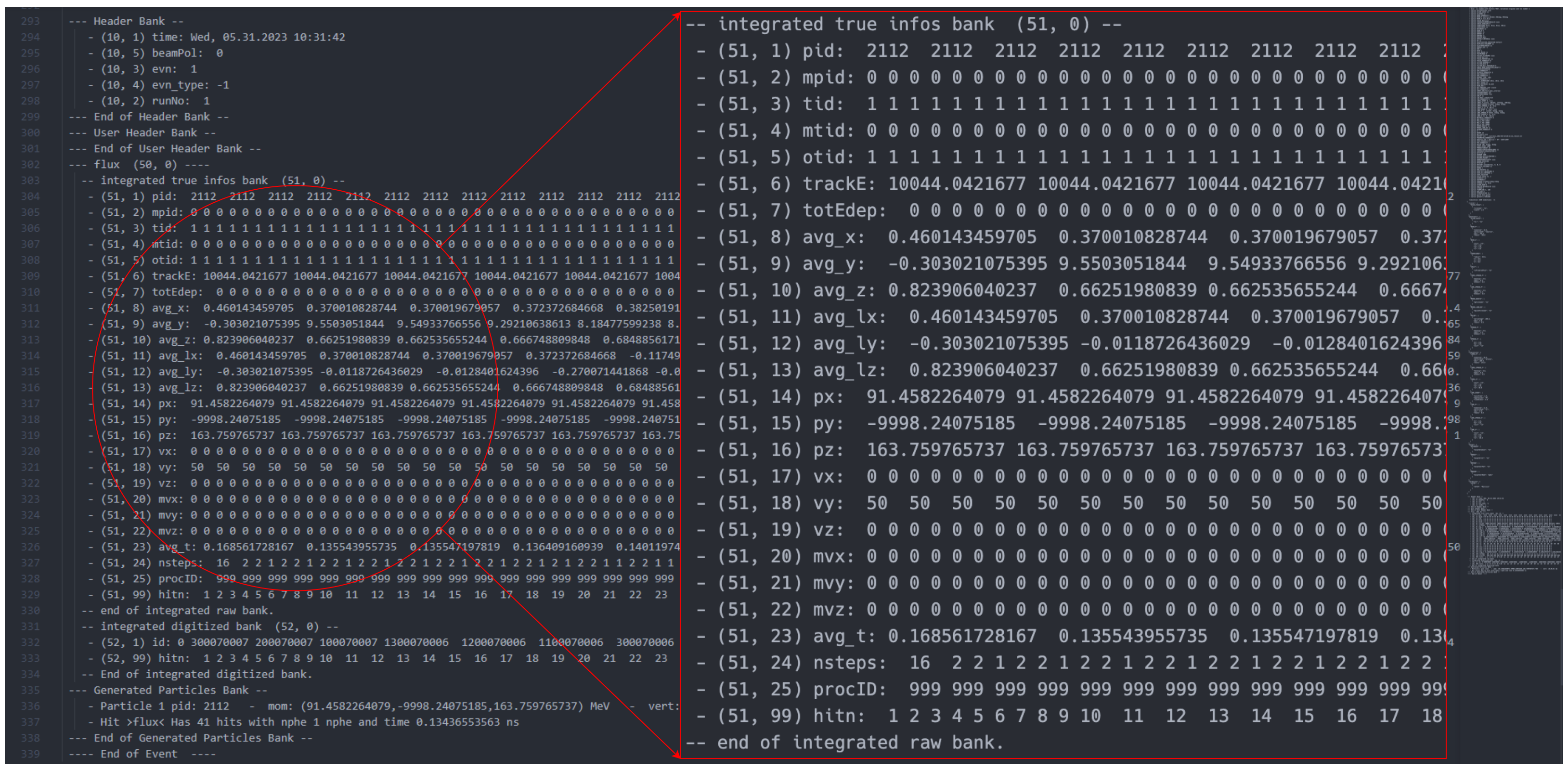
```
<gcard>
    <detector name="bcal" factory="TEXT" variation="original"/>
    <option name="INTEGRATEDRAW" value="flux"/>
    <option name="SAVE_ALL_MOTHERS" value="1"/>
    <option name="PHYSICS" value="STD + Optical + HP + QGSP_BERT"/>
    <option name="RECORD_OPTICALPHOTONS" value="1"/>
    <option name="RECORD_PASSBY" value="1"/>
    <option name="BEAM_P" value="mu-,10*GeV,-90*deg,-90*deg"/>
    <option name="BEAM_V" value="(0,5,0)cm"/>
    <option name="SPREAD_P" value="0*GeV,3*deg,3*deg"/>
    <option name="N" value="1"/>
    <option name="USE_GUI" value="0"/>
    <option name="OUTPUT" value="txt, ../out/output.txt"/>
</gcard>
```

Figure 18: GCARD example for muon particle collisions. `Source: Own elaboration`

It is important to note that this whole part of the process is executed in a parallel fashion. Jobs are sent to the CCTVal Cluster's queue to generate simulations and are automatically distributed across the available nodes. This allowed for an approximate generating rate of 917 simulations per hour, for a data set the size of about 10.000 entries.

3.2.3 GEMC OUTPUT HANDLING

GEMC's output handling is straightforward. The generated output files are grouped into a particle-specific folder. Then, they are processed individually using the Gruid Translator to convert them from EVIO to a Python sparse matrix format. Figure 19 displays a snapshot of the EVIO-formatted output from a very small simulation.



```
--- Header Bank --
    - (10, 1) time: Wed, 05.31.2023 10:31:42
    - (10, 5) beamPol:  0
    - (10, 3) evn:  1
    - (10, 4) evn_type: -1
    - (10, 2) runNo:  1
--- End of Header Bank --
--- User Header Bank --
--- End of User Header Bank --
--- flux  (50, 0) ----
  -- integrated true infos bank  (51, 0) --
   - (51, 1) pid:  2112  2112  2112  2112  2112  2112  2112  2112  2112  2112  2112  2112  2112
   - (51, 2) mpid: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 3) tid:  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   - (51, 4) mtid: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 5) otid: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   - (51, 6) trackE: 10044.0421677 10044.0421677 10044.0421677 10044.0421677 10044.0421677 1004
   - (51, 7) totEdep:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 8) avg_x:  0.460143459705  0.370010828744  0.370019679057  0.372372684668  0.38250191
   - (51, 9) avg_y:  -0.303021075395 9.5503051844  9.54933766556 9.29210638613 8.18477599238 8.
   - (51, 10) avg_z: 0.823906040237  0.66251980839 0.662535655244  0.666748809848  0.6848856171
   - (51, 11) avg_lx:  0.460143459705  0.370010828744  0.370019679057  0.372372684668  -0.11749
   - (51, 12) avg_ly:  -0.303021075395 -0.0118726436029  -0.0128401624396  -0.270071441868 -0.0
   - (51, 13) avg_lz:  0.823906040237  0.66251980839 0.662535655244  0.666748809848  0.68488561
   - (51, 14) px:  91.4582264079 91.4582264079 91.4582264079 91.4582264079 91.4582264079 91.458
   - (51, 15) py:  -9998.24075185  -9998.24075185  -9998.24075185  -9998.24075185  -9998.240751
   - (51, 16) pz:  163.759765737 163.759765737 163.759765737 163.759765737 163.759765737 163.75
   - (51, 17) vx:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 18) vy:  50  50  50  50  50  50  50  50  50  50  50  50  50  50  50  50  50  50  50
   - (51, 19) vz:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 20) mvx: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 21) mvy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 22) mvz: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   - (51, 23) avg_t: 0.168561728167  0.135543955735  0.135547197819  0.136409160939  0.14011974
   - (51, 24) nsteps:  16  2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 1 2 2 1 1 2 2 1 1
   - (51, 25) procID:  999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
   - (51, 99) hitn:  1 2 3 4 5 6 7 8 9 10  11  12  13  14  15  16  17  18  19  20  21  22  23
  -- end of integrated raw bank.
  -- integrated digitized bank  (52, 0) --
   - (52, 1) id: 0 300070007 200070007 100070007 1300070006  1200070006  1100070006  300070006
   - (52, 99) hitn:  1 2 3 4 5 6 7 8 9 10  11  12  13  14  15  16  17  18  19  20  21  22  23
  -- End of integrated digitized bank.
--- Generated Particles Bank --
   - Particle 1 pid: 2112   -  mom: (91.4582264079,-9998.24075185,163.759765737) MeV   -  vert:
   - Hit >flux< Has 41 hits with nphe 1 nphe and time 0.13436553563 ns
--- End of Generated Particles Bank --
---- End of Event  ----
```

```
-- integrated true infos bank  (51, 0) --
 - (51, 1) pid:  2112  2112  2112  2112  2112  2112  2112  2112  2112
 - (51, 2) mpid: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 3) tid:  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 - (51, 4) mtid: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 5) otid: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 - (51, 6) trackE: 10044.0421677 10044.0421677 10044.0421677 10044.0421
 - (51, 7) totEdep:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 8) avg_x:  0.460143459705  0.370010828744  0.370019679057  0.37
 - (51, 9) avg_y:  -0.303021075395 9.5503051844  9.54933766556 9.292106
 - (51, 10) avg_z: 0.823906040237  0.66251980839 0.662535655244  0.6667
 - (51, 11) avg_lx:  0.460143459705  0.370010828744  0.370019679057  0.
 - (51, 12) avg_ly:  -0.303021075395 -0.0118726436029  -0.0128401624396
 - (51, 13) avg_lz:  0.823906040237  0.66251980839 0.662535655244  0.66
 - (51, 14) px:  91.4582264079 91.4582264079 91.4582264079 91.458226407
 - (51, 15) py:  -9998.24075185  -9998.24075185  -9998.24075185  -9998.
 - (51, 16) pz:  163.759765737 163.759765737 163.759765737 163.75976573
 - (51, 17) vx:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 18) vy:  50  50  50  50  50  50  50  50  50  50  50  50  50  50
 - (51, 19) vz:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 20) mvx: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 21) mvy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 22) mvz: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 - (51, 23) avg_t: 0.168561728167  0.135543955735  0.135547197819  0.13
 - (51, 24) nsteps:  16  2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2
 - (51, 25) procID:  999 999 999 999 999 999 999 999 999 999 999 999 99
 - (51, 99) hitn:  1 2 3 4 5 6 7 8 9 10  11  12  13  14  15  16  17  18
-- end of integrated raw bank.
```

Figure 19: GEMC EVIO Output Format, `Source: Own elaboration`

The data set is built on the premise that every simulation file generated contains only one event. This is for two reasons. First, the fact that the data is intended to be used to train a machine learning based method, so as said data is passed to the machine, there's the need to capture the signal generated by only one cosmic ray collision, since the data is meant to be as granular as possible. Second, future work on potential time-series analysis for e.g. anomaly detection, will be simplified as every file will have it's own timestamp, allowing for automated data labeling as well.

3.3 TRANSLATION

An important limitation arises in this stage of the process. The Gruid Translator is capable of translating simulated collisions involving muon and antimuon particles only. This is because standard detectors, like the BabyCal, are usually expected to detect muon particles, since they are commonly used for the study of the physical phenomena of time dilation and length contraction which are demonstrations for the theory of general relativity [Bailey et al., 1979], [Behroozi, 2014]. This narrows down the amount of particles for The Framework to process, but it's a very good starting point since it's the most "*similar to real life*" case to consider in this thesis work.

3.3.1 GRUID TRANSLATOR INPUT HANDLING

Like the previous part of the process, Gruid's input handling is also a straightforward process. The whole particle-specific folder with the simulated EVIO output textfiles is taken and is passed on to Gruid obtaining a folder with the translations of every simulated file.

Gruid's positional arguments are four:

1. `filename`: Path of the gemc file to be processed.
2. `dt`: The length of each time step for the generated time series in ns.
3. `dx`: The length of each row for each of the time series' matrices in cm.
4. `dy`: The length of each column for each of the time series' matrices in cm.

The arguments `dt`, `dx` and `dy` will get a more detailed explanation:

As described previously, the BabyCal's scintillating fibers will light up creating a "shower image", in this case a matrix, on both faces of the detector (one in each side). Considering the total time of the simulation, `dt` allows the user to get a time series of different matrices with the progression of how the final showers were generated in each face of the detector. For this thesis work, the emphasis will be put on analyzing the final shower image, since the objective is to determine conclusions on the spatial behaviour of the data. For that to be achieved, a high enough `dt` value will be chosen to omit the time series and get a matrix with two sides. This can be also seen as a 2-dimensional image with two channels (side 1 and side 2).

For `dx` and `dy`, an imaginary pair of grids can be placed on each face of the detector. In other words as the ones that describe the meaning of each value, they can also be seen as the height and width of each coordinate of the matrix. When taking this to a more graphical level for understanding, they represent the dimensions of each pixel of the image that will be generated in each side of the detector, meaning, they define the resolution of the image. For the **simulated** BabyCal's specific case, one fiber of the detector has 0.1 centimeters of diameter; so the base resolution (the highest possible), that makes sense to consider would be values of 0.1 cm for both `dx` and `dy`. As that would allow to capture one fiber per pixel. For a definitive understanding of this concept, Figure 20 shows an example for one side of a simulated BabyCal with Gruid's grid on it and the values of `dx` and `dy` presented graphically.

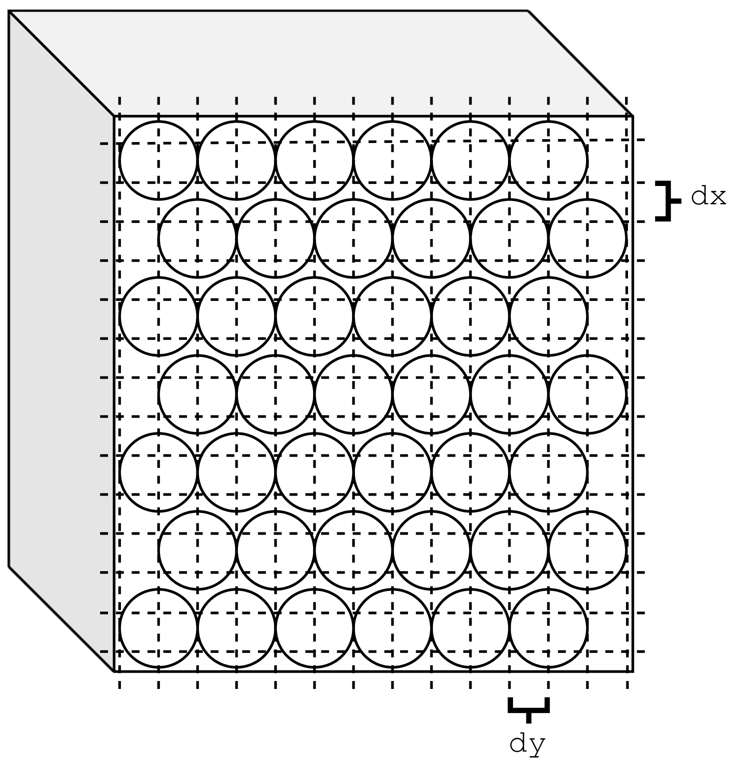


Figure 20: BabyCal vs Gruid Translator's Grid. `Source: Own elaboration`

For its use, Gruid has an OUTTYPE input option (see Table 3), which let's the user choose the format of the output, allowing for a better and easier use of the generated data. If the user wants to choose the data output format, the use of that flag in the input for Gruid's execution is mandatory.

The Value of `OUTTYPE` must be a number and can vary from `1` to `5`:

1. Print the generated `.json` to `stdout`, mainly for testing. This option can generate a very extensive amount of output data, so it should be used with caution.
2. A `.json` file containing the time series is stored in `out/`. This is the default option.
3. The massive particle hits are added to the exported file. These are the targets for reconstruction.
4. The photon hits used to generate the time series are added to the exported file. This can aid in debugging.
5. The metadata taken from the GEMC input file is added to the `.json` file. This can be useful if the user wants to destroy that file or extract some information from it.

Table 3: Gruid Translator Arguments Table, `Source:` Gruid Translator Docs

Argument	-	**Description**
`filename`	-	Path of the gemc file to be processed.
`dt`	-	Length of each time step for the generated time series in ns.
`dx`	-	Length of each row for each of the time series' matrices in cm.
`dy`	-	Length of each column for each of the time series' matrices in cm.
Optional Arguments		
`-h, --help`	-	Show this help message and exit.
`-z DZ, --dz DZ`	-	Length of each depth column for each of the detector's body time series' matrices in cm.
`--pvx PVX`	-	x position of the vertex for the detecting plane inside the detector's body.
`--pvy PVY`	-	y position of the vertex for the detecting plane inside the detector's body.
`--pvz PVZ`	-	z position of the vertex for the detecting plane inside the detector's body.
`--pnx PNX`	-	x direction of the normal vector to the detecting plane inside the detector's body.
`--pny PNY`	-	y direction of the normal vector to the detecting plane inside the detector's body.
`--pnz PNZ`	-	z direction of the normal vector to the detecting plane inside the detector's body.
`-f FEVENT, --fevent FEVENT`	-	Event number of the first event in the gemc file that should be read. Note that events are counted from 1 onward. Default is 1.
`-n NEVENTS, --nevents NEVENTS`	-	Number of events to read, counting from the file set with FEVENT. Set to 0 to read until the end of file. Default is 0.
`-o OUTTYPE, --outtype OUTTYPE`	-	Type of output to be generated. Can be any integer from 1 to 5. Check the README for a detailed description of each alternative. Default is 2.
`-r NROWS, --nrows NROWS`	-	Number of rows set in the gemc simulation. By default this is read from the filename, but this argument can be set to override this behaviour.
`-c NCOLS, --ncols NCOLS`	-	Number of columns set in the gemc simulation. By default this is read from the filename, but this argument can be set to override this behaviour.

An example Python script of how to program a batch translation with Gruid is shown in Figure 21. The example shows the use of all mandatory positional arguments, being `dt = 5 ns`, `dx = 0.1 cm` and `dy = 0.1 cm`, omitting the `-o OUTTYPE` option, since all is needed for this case is the default Gruid output format.

```python
import os

# Access the script's directory (cd /user/d/dhebel/babycal/gruid-
    translator/)
os.chdir("/user/d/dhebel/babycal/gruid-translator/")

# Iterate over all files in "sims/mu-" and translate them applying
    the gruid-translator
for FNAME in os.listdir("/user/d/dhebel/babycal/sims/mu-"):
    os.system('bash run.sh /user/d/dhebel/babycal/sims/mu-/{} 5 0.1
        0.1'.format(FNAME))
    os.system('echo "Gruid translation finished for {}"'.format(
        FNAME))

# Now for mu+:
for FNAME in os.listdir("/user/d/dhebel/babycal/sims/mu+"):
    os.system('bash run.sh /user/d/dhebel/babycal/sims/mu+/{} 5 0.1
        0.1'.format(FNAME))
    os.system('echo "Gruid translation finished for {}"'.format(
        FNAME))
```

Figure 21: Gruid Translation Script, `Source: Own elaboration`

3.4 DATA PRE-PROCESSING

Gruid's output for this use case is a dictionary of 2-dimensional sparse matrices. Each matrix is defined as a dictionary where a key is a *tuple-like* string with the position coordinates in the matrix and their respective value is the energy deposited in MeV. The code snippet shown in Figure 22 is an example of Gruid's default output format.

```
{
    "output.txt event 1": {
        "gruid hits - side 1": {
            "0.0": {
                "4,0": {
                    "# of hits": 7,
                    "energy deposited": 4.5034438176719995e-05
                },
                "4,1": {
                    "# of hits": 2,
                    "energy deposited": 1.635993026225e-05
                },
                "4,2": {
                    "# of hits": 6,
                    "energy deposited": 3.519082653253e-05
                },
                ...
            }
        },
        "gruid hits - side 2": {
            "0.0": {
                "4,0": {
                    "# of hits": 5,
                    "energy deposited": 3.747209241649e-05
                },
                ...
            }
        },
        "gruid metadata": {
            "# of columns (x)": 5,
            "# of rows (y)": 7,
            "dt": 5.0,
            "dx": 0.3,
            "dy": 0.3,
            "pid": 13
        }
    }
}
```

Figure 22: Gruid output example (`OUTTYPE: 2`). `Source: Own elaboration`

The whole folder is processed and the information of each file is appended to a numpy matrix (the matrix gets reconstructed into the earlier mentioned pythonic format). Two matrices (`X` and `y`) are generated for each resolution chosen for experimentation and testing of the output data. Matrix `X` will represent the dataset with the energy in **MeV** for every sparse coordinate of the matrix and `y` will represent the target vector for every simulated event. Since the intention is now to analyze muons and antimuons only due to Gruid's limitation to process those particles exclusively, the target vectors' values will be defined as the following:

- `1`: muons
- `0`: antimuons

It is important to remember that in the context of this thesis work, events and simulations mean the same, since every simulation has only one event. From now on, they will also be referred as the same indistinctly.

To clarify this concept in a graphical way, Figure 23 shows the output structure of the reconstructed matrices.

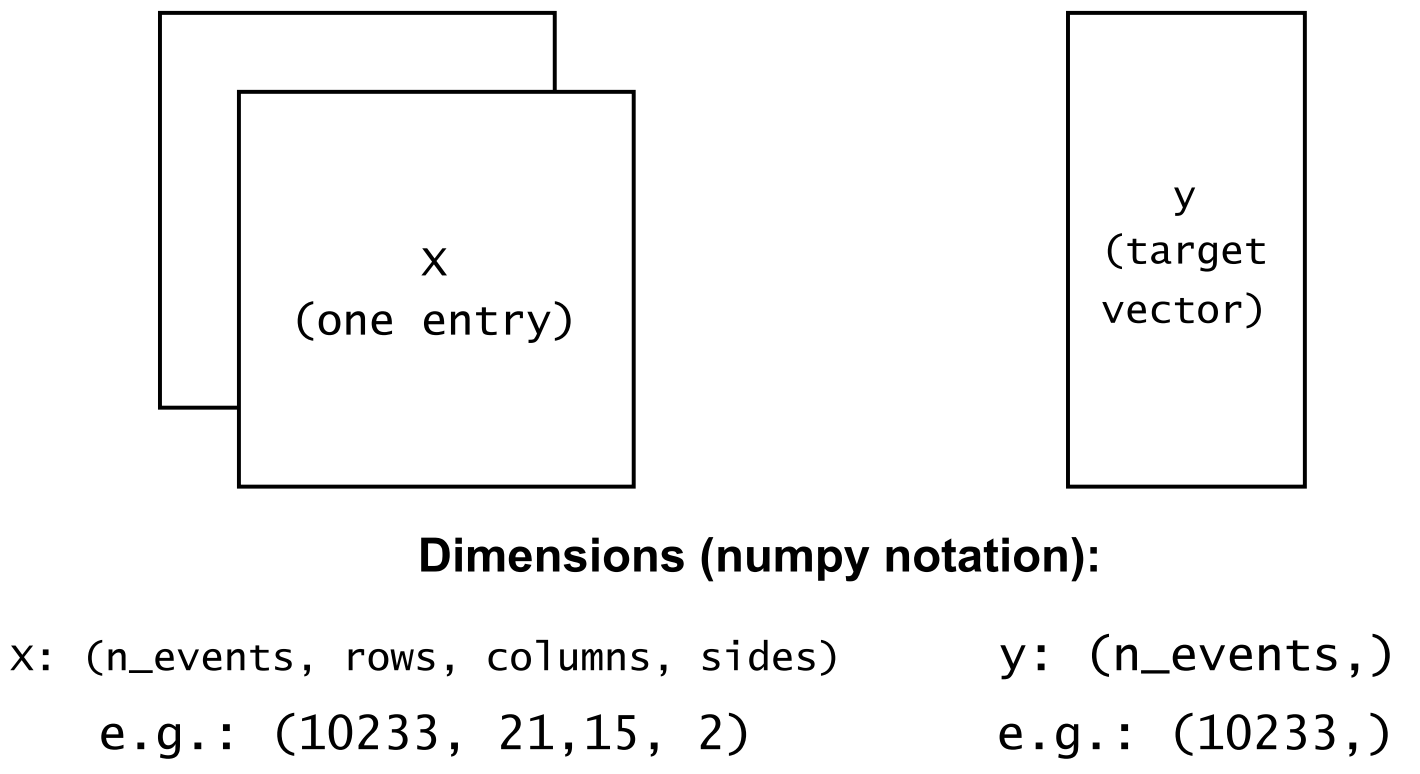


Figure 23: Example of Gruid's output numpy matrix dimensions for a `dx=dy=0.1` resolution after reconstruction. The number of rows and columns will be 21 and 15 respectively since the considered resolution is the size of each fiber. Gruid adds 6 (in this case) extra rows for spacing purposes. `Source: Own elaboration`

The following code snippet (Figure 24) is the function that receives a list of json files (the default output format shown earlier), and builds the exampled dataset (10.233 entries) reconstructing every json file into the numpy matrix format and appending it to the `X` matrix.

```python
def reconstruct_matrix(filelist):
    X = []

    for file in filelist:
        with open(file) as f:
            data = json.load(f)

        for event, event_data in data.items():
            side_1 = event_data['gruid hits - side 1']
            side_2 = event_data['gruid hits - side 2']
            gruid_metadata = event_data['gruid metadata']

        matrix = np.zeros((gruid_metadata['# of rows (y)'], gruid_metadata['# of columns (x)'], 2))

        # IMPORTANT: Coordinates are inverted in the matrix because of matrix notation in GRUID.

        # Side 1
        for timestamp, timestamp_data in side_1.items():
            for pixel, pixel_data in timestamp_data.items():
                x, y = pixel.split(",")
                matrix[ int(y), int(x), 0 ] = pixel_data['energy deposited']

        # Side 2
        for timestamp, timestamp_data in side_2.items():
            for pixel, pixel_data in timestamp_data.items():
                x, y = pixel.split(",")
                matrix[ int(y), int(x), 1 ] = pixel_data['energy deposited']

        X.append(matrix)

    return np.array(X)

muon_labels = np.ones(len(files_MUM_01))
antimuon_labels = np.zeros(len(files_MUP_01))
y_01 = np.concatenate((muon_labels, antimuon_labels))

muons = reconstruct_matrix(files_MUM_01)
antimuons = reconstruct_matrix(files_MUP_01)
X_01 = np.concatenate((muons, antimuons))
```

Figure 24: Matrix reconstruction script for the Gruid default output format. Source: Own elaboration

CHAPTER 4
MACHINE LEARNING FOR DATA ANALYSIS

4.1 CNN FOR CLASSIFICATION: SVGG-21Net

At this point, we have generated simulated data using GEMC and Gruid, which is labeled and has the proper format to be used by neural network Python modules. Thus, making the format ideal for using Convolutional Neuronal Networks (CNNs), a supervised machine learning method, to classify and distinguish muons from antimuons. That makes the task of classifying particles a very interesting one to execute, so it was the first automatic analysis approach added to The Framework.

Prior to feeding the data to the CNN, the matrices were pre-processed and padded with zeros into squared images, so the dimensions are a better fit with the kernel size and its strides, see Figure 25. Various kernel sizes and strides where tested, but the parameters given in Table 4 were the perfect fit, they were the only sizes that worked with the matrices dimensions.

The exampled dataset used in the Gruid's output matrix dimensions diagram are the same ones used for the experimentation. The matrices dimensions are brought from (21, 15, 2) to (21, 21, 2); see Figure 25.

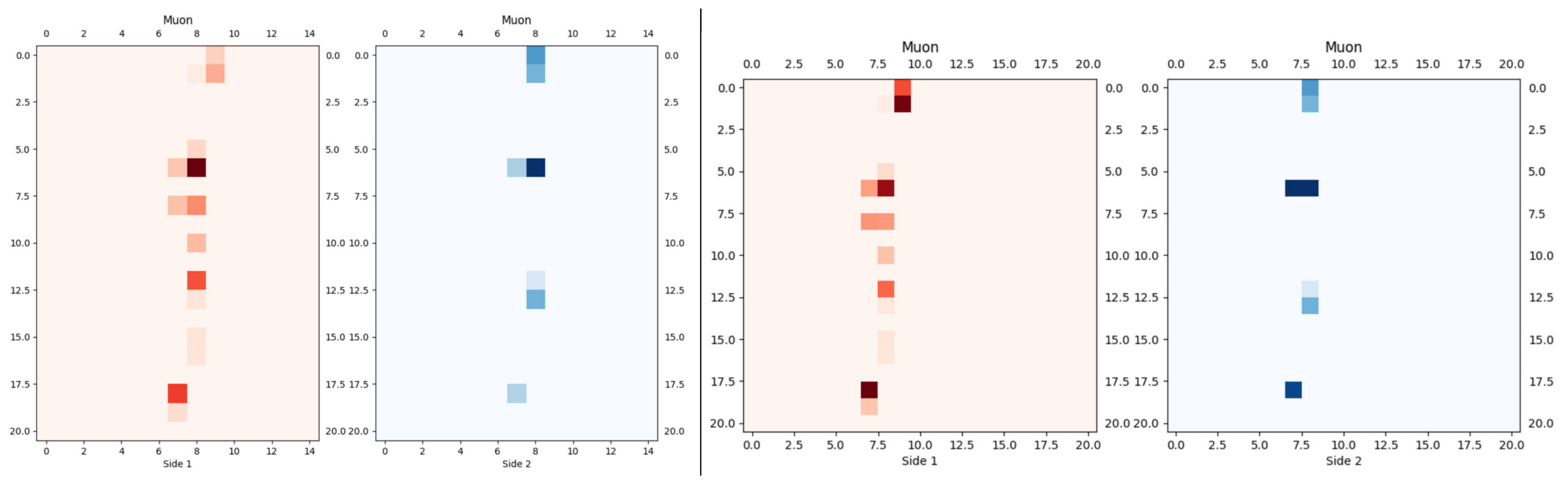


Figure 25: Particle Plots: Pre-Padding Vs Post-Padding (Muon Example)

The proposed CNN architecture is a simplified variant for the standard VGGNet architecture type. It consists of several layers, including convolutional layers, max-pooling layers, and fully connected layers [Simonyan and Zisserman, 2015].

The following table (see Table 4) describes the result for the experiment alongside the CNN's architecture. The plots correspond to the performance metrics progressions throughout the epochs (see Figure 26 and 27).

Table 4: SVGG-21Net Architecture Detail and Results

Layer	**Configuration**		
Input	(Batch Size, Width, Height, Channels)		
Conv2D (32 filters)	Kernel Size: (3, 3), Strides: (1, 1), Activation: ReLU		
Conv2D (64 filters)	Kernel Size: (3, 3), Strides: (1, 1), Activation: ReLU		
MaxPooling2D	Pool Size: (2, 2), Strides: (2, 2)		
Conv2D (128 filters)	Kernel Size: (3, 3), Strides: (1, 1), Activation: ReLU		
MaxPooling2D	Pool Size: (2, 2), Strides: (2, 2)		
Flatten			
Dense (128 units)	Activation: ReLU		
Dense (2 units)	Activation: Softmax		
Training Details	**Values**		
Batch Size	Default		
Number of Epochs	30		
Optimizer	Adam		
Learning Rate	Default		
Loss Function	Categorical Cross-entropy		
Dataset Details	**Values**		
Dataset Name	mump_0.1		
Number of Classes	2		
Dataset Size	10.233		
Data Preprocessing	Normalized with `keras.utils.normalize()`		
Target Encoding	One-Hot Encoding		
Used Performance Metrics			
Accuracy	Recall		
Precision	F1-Score		
Results	**Scores**	**Results**	**Scores**
Training Loss	0.0281	Training Accuracy	0.9916
Validation Loss	3.3435	Validation Accuracy	0.5366
Testing Loss	3.6665	Testing Accuracy	0.5164
Training Precision	0.9916	Training Recall	0.9916
Validation Precision	0.5366	Validation Recall	0.5366
Testing Precision	0.5164	Testing Recall	0.5164
Training F1-Score	0.4941		
Validation F1-Score	0.2822	Training Time	135.71 seconds
Testing F1-Score	0.2718		

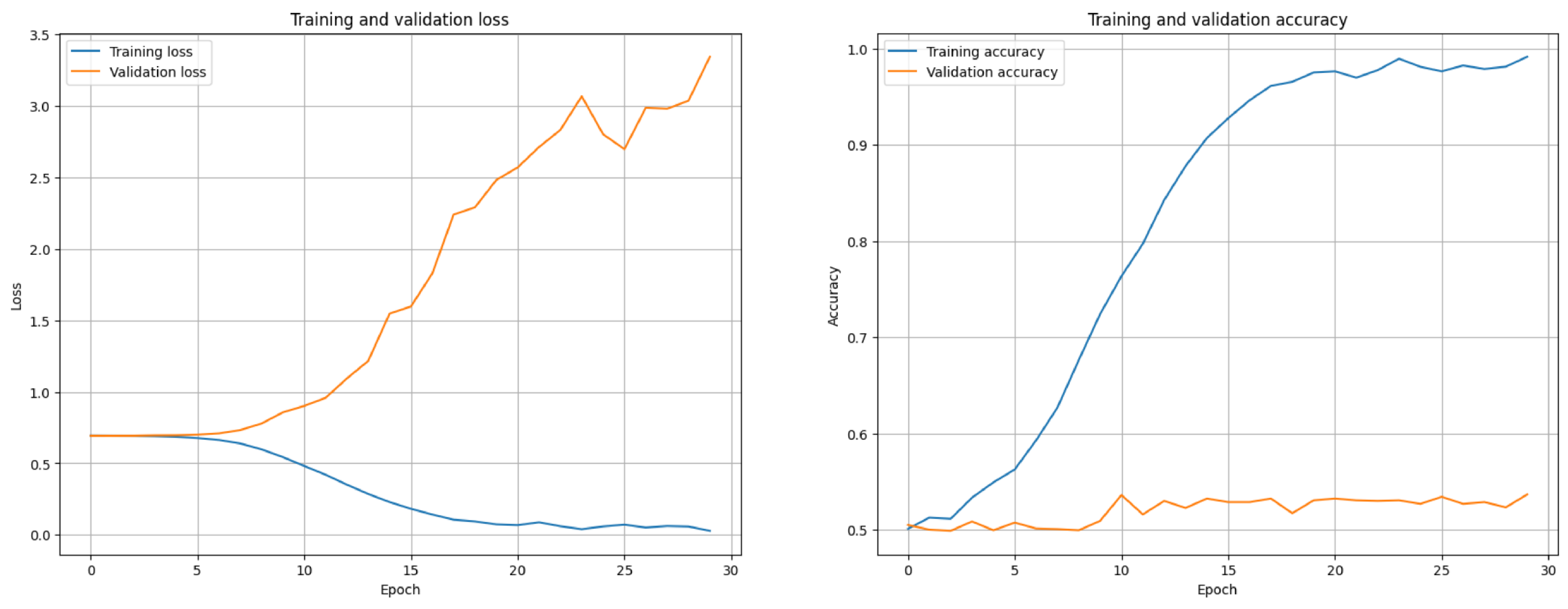


Figure 26: SVGG-21Net: Loss & Accuracy Behaviour

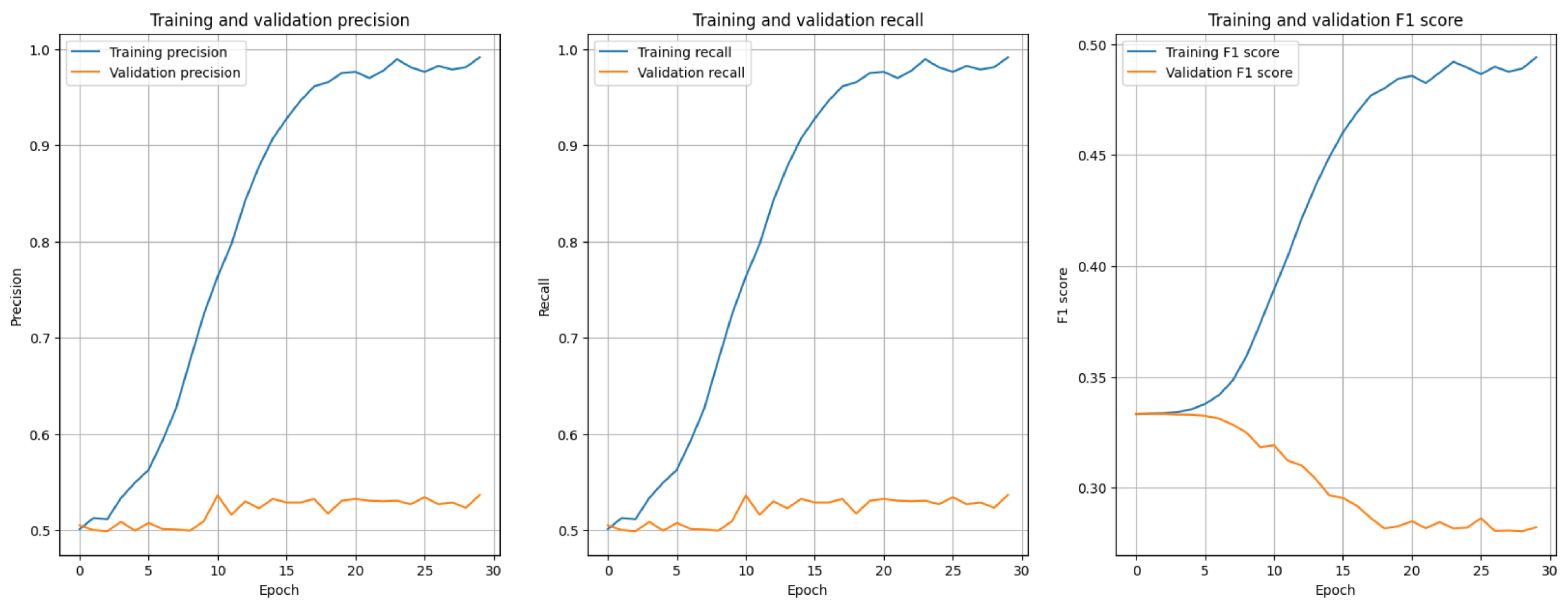


Figure 27: SVGG-21Net: Precision, Recall & F1-Score Behaviour

The CNN's training performance demonstrated high accuracy and recall. However, discrepancies arise when evaluating on the validation and testing sets. The elevated loss and reduced accuracy on the validation set indicate potential overfitting, where the model's performance on unseen data is deficient.

To illustrate the summary of the performance metrics, we'll visualize the confusion matrix on the testing set, see Figure 28.

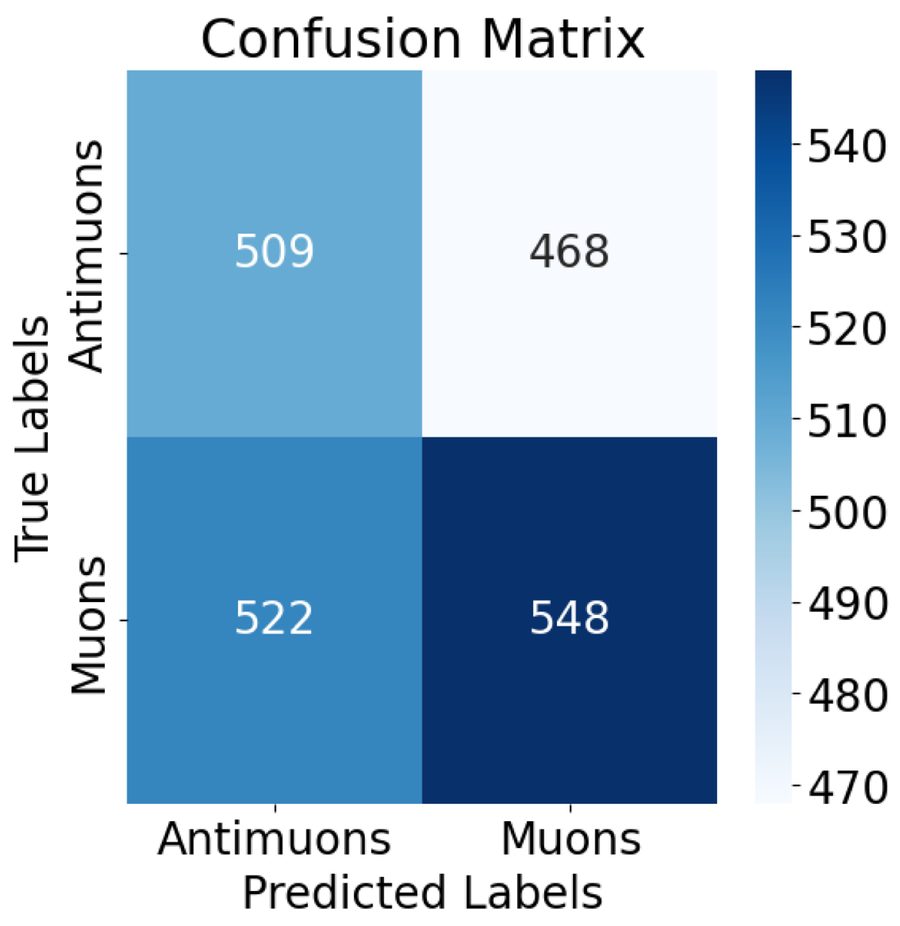


Figure 28: SVGG-21Net: Confusion matrix

Its behaviour shows a distinctive type of overfitting, where the model mistakenly believes it is learning, specially in the beginning of the training stage. However, during validation with loss, accuracy and the rest of the performance metrics (see Section 2.5.4), it diverges significantly from the expected curves. This strongly suggests that the classes in the dataset lack clear separability from one another. The testing results confirm the deficient performance of the CNN when trying to classify the particles.

4.2 AUTOENCODER FOR DATA RECONSTRUCTION: AE-21RecNet

Due to the suggested lack of separability of the classes when trying to classify with the SVGG-21Net CNN, an autoencoder (AE-21RecNet) was trained exclusively with muon data to test reconstruction with muons. The proposed architecture corresponds to Table 5. The plots correspond to the performance metrics progressions throughout the epochs; see Figure 29.

Table 5: AE-21RecNet Architecture and Training Details (dx=dy=0.1)

Layer	**Configuration**
Input	(Batch Size, Width, Height, Channels)
Flatten	
Dense (630 units)	Units: Width x Height x Channels, Activation: ReLU
Dense (315 units)	Units: Width x Height, Activation: ReLU
Dense (21 units)	Units: Width, Activation: ReLU
Dense (21 units)	Units: Width, Activation: ReLU
Dense (315 units)	Units: Width x Height, Activation: ReLU
Dense (630 units)	Units: Width x Height x Channels, Activation: ReLU
Reshape	Shape: (Width, Height, Channels)
Training Details	**Values**
Optimizer	Adam
Loss Function	Mean Squared Error (MSE)
Number of Epochs	30
Batch Size	Default
Dataset Details	**Values**
Dataset Name	mum_0.1
Number of Samples	5.214
Data Preprocessing	Normalized with `keras.utils.normalize()`
Used Performance Metrics	
AE Accuracy	
Results	**Scores**
Training Loss	0.0022
Validation Loss	0.0033
Testing Loss	0.0035
Training AE Accuracy	0.9803
Validation AE Accuracy	0.9756
Testing AE Accuracy	0.9743
Training Time	36.48 seconds

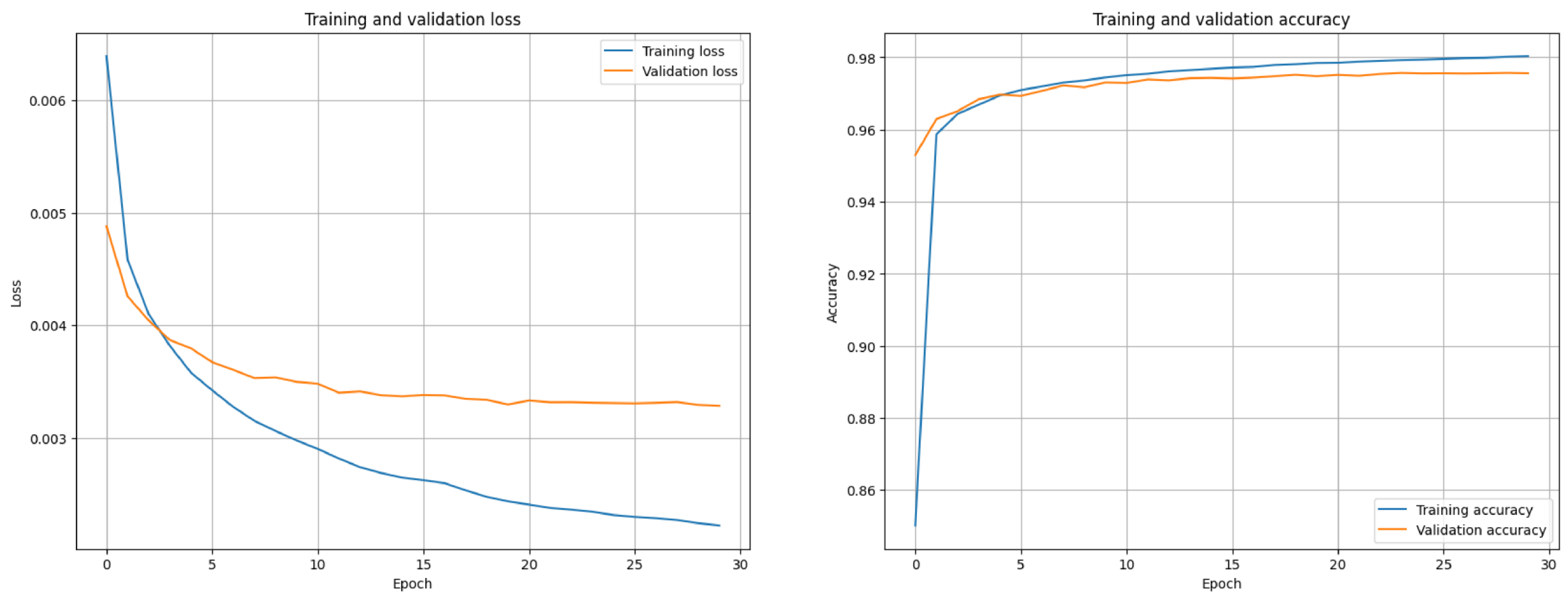


Figure 29: AE-21RecNet: Loss & Accuracy Behaviour

In the context of autoencoders, accuracy is not interpreted the same way it's done in convolutional neural networks, because it's strictly meaningless to ask for the accuracy from an autoencoder. Instead, autoencoders usually use the loss function to describe the performance of the model. Nevertheless, it is important to note that, since the autoencoder trained for this reconstruction task used the Keras Python library[10], it allowed an accuracy function to be used for further analysis and, mainly, visualization purposes. Which in this case, since the model is meant for reconstruction, is calculated by dividing the number of correctly reconstructed events by the number of total events [Keras Team, 2023] meaning, it is calculated just like regular accuracy in a CNN, but it is to be interpreted in a different way.

$$\text{AE Accuracy (}\textbf{Auto}\textbf{E}\text{ncoder Accuracy)} = \frac{\text{Correctly Reconstructed Events}}{\text{Total Number of Events}}$$

Since autoencoders aim to learn an efficient representation of the input data, the accuracy metric can provide a measure of how accurately the model can reconstruct the input samples. Higher accuracy values indicate better reconstruction performance.

This behaviour (see Figure 29) indicates a nice fit for the learning curves. The loss function (MSE) plot shows a consistent decrease in the loss value during training, indicating that the autoencoder is effectively learning to reconstruct the data. A decreasing loss value is a positive sign, indicating improved performance and convergence throughout the epochs. The Accuracy plot shows a stable or gradually improving trend. While not a traditional accuracy measure, an increasing trend in the accuracy plot means that the autoencoder successfully reconstructs the data with greater fidelity. If the loss has stabilized at a low value and the accuracy shows improvement, it suggests that the autoencoder has reached a satisfactory reconstruction capability.

[10]Keras Python Library

The MSE Error Reconstruction is usually the *"go-to"* loss function when performing reconstruction tasks with autoencoders. It can be visualized trough the MSE error reconstruction plot to see the behaviour of the loss function troughout the events. This plot was generated to visualize muon reconstruction. After promising results, antimuon reconstruction was also added to the plot in order to compare with the muon reconstruction results. (see Figure 30).

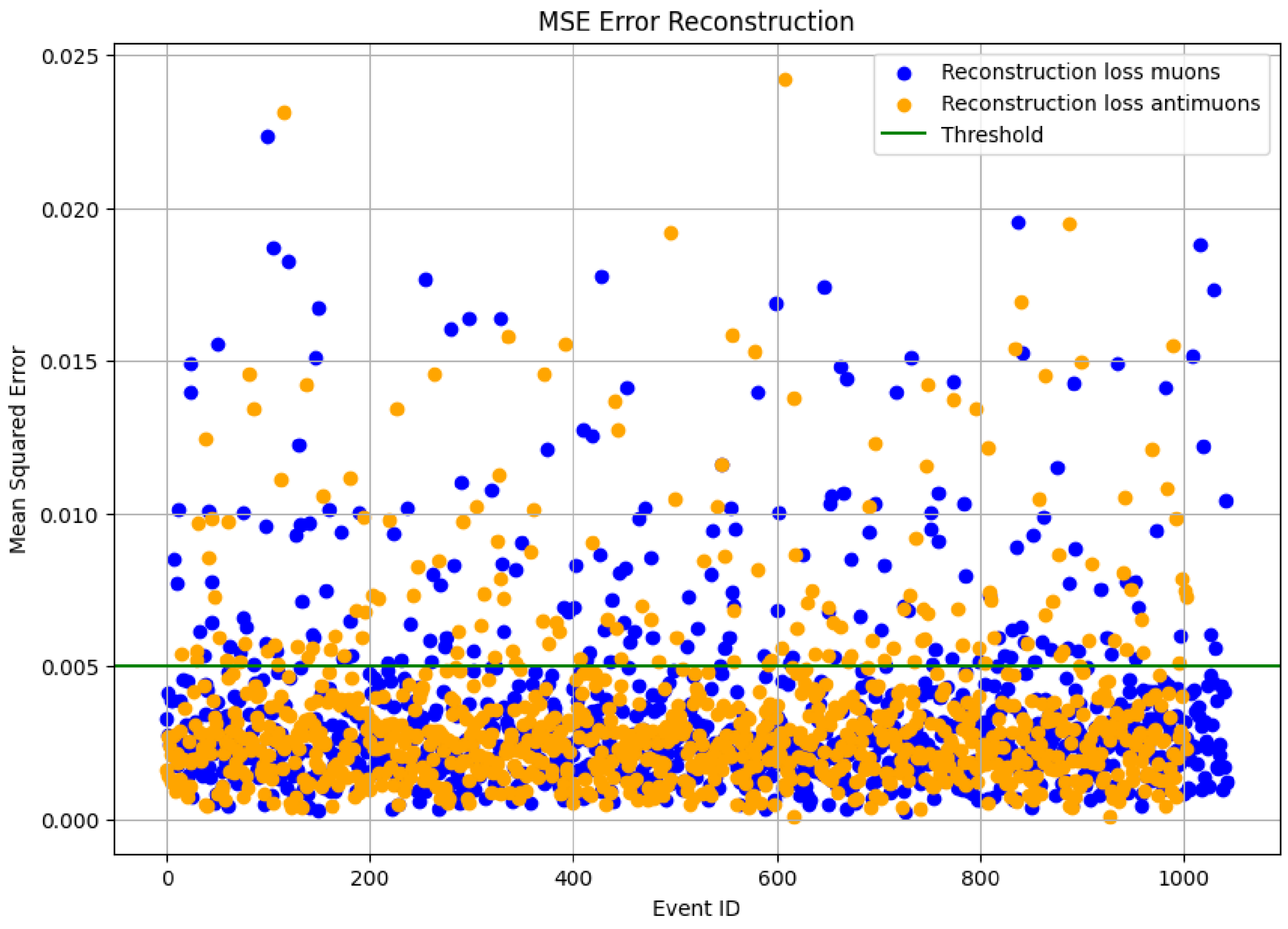


Figure 30: MSE Error Reconstruction

The MSE plot exhibits the MSE values for reconstruction with both muons and antimuons. The plot demonstrates minimal differences between the MSE values of these particle's reconstructions. Essentially, the AE-21RecNet autoencoder performs similarly for muons and antimuons.

A lower MSE value indicates better reconstruction performance. A threshold was set at 0.005 to identify when data deviates from the main cluster of MSE behavior. However, the number of events exceeding this threshold is very limited compared to the rest.

It shows there's no noticeable clustered outliers when it comes to the MSE loss function. No anomalies were observed in the MSE plot, but the similarity in the results between muons and antimuons suggests the need for further analysis.

When testing graphical reconstruction with the AE-21RecNet autoencoder, it's clear to see that antimuons can be reconstructed just as accurately as muons even when the autoencoder is trained exclusively on muon data, see Figure 31 and 32. This finding suggests that, within the given dataset and representation, muons and antimuons lack distinct spatial features, posing challenges for their classification. In other words, the autoencoder sees both particles like they are the same.

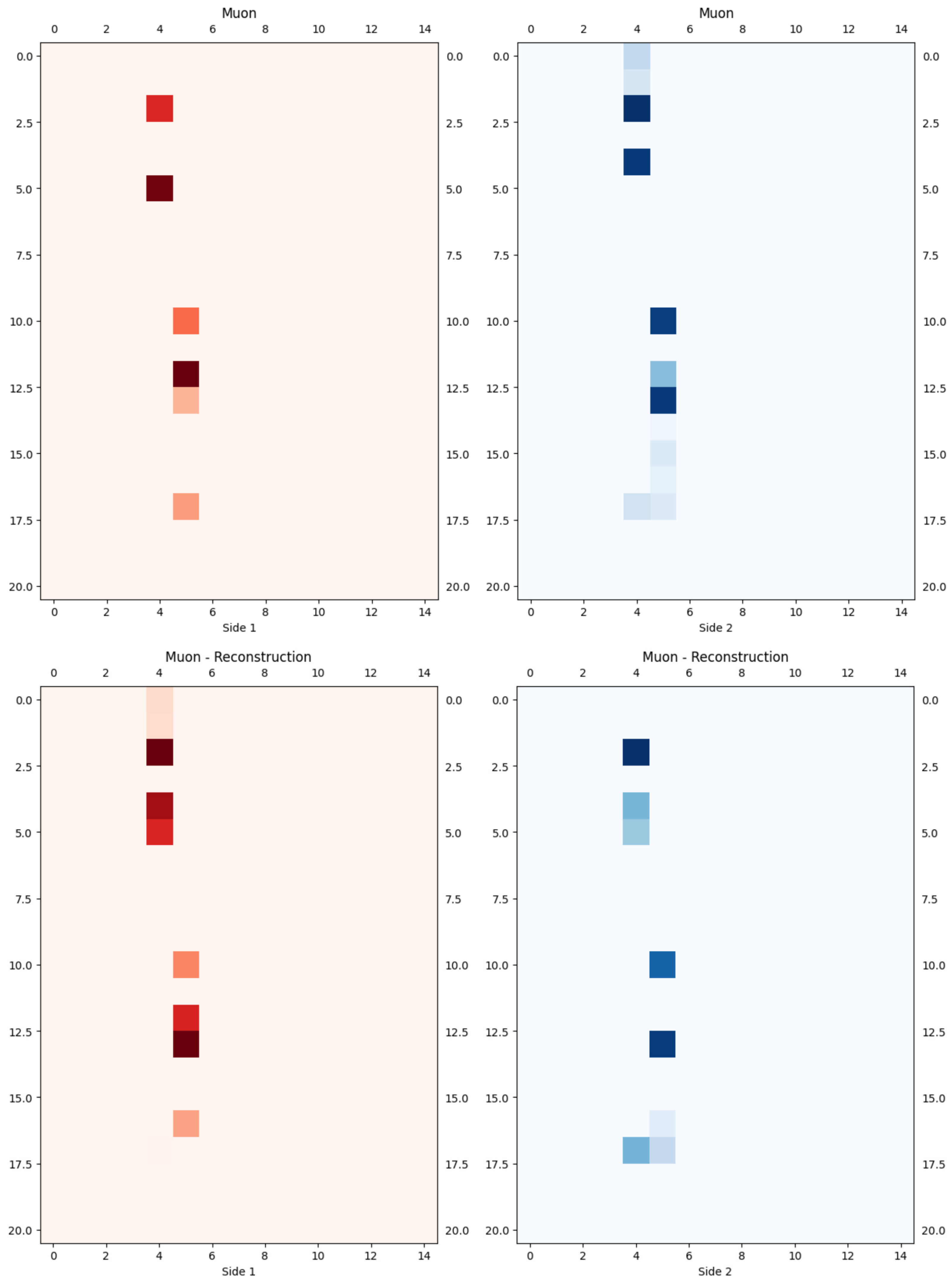


Figure 31: AE-21RecNet: Muon vs Muon Reconstructions. Top row shows the actual muon, and bottom row shows the reconstructed muon using the autoencoder.

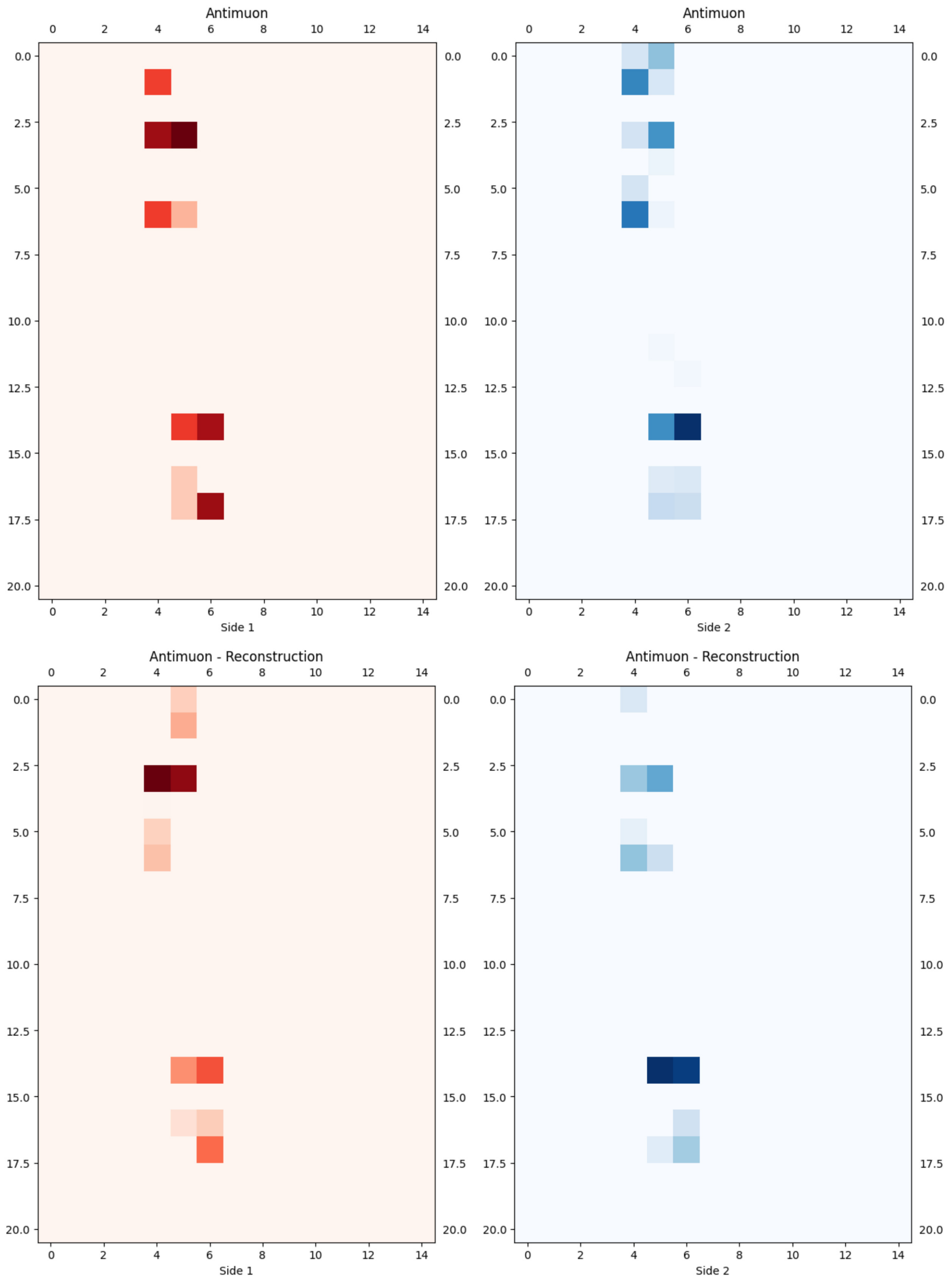


Figure 32: AE-21RecNet: Antimuon vs Antimuon Reconstructions. Top row shows the actual antimuon, and bottom row shows the reconstructed antimuon using the autoencoder.

CHAPTER 5
CONCLUSIONS

5.1 GENERAL CONCLUSIONS

This thesis work presented an automated system capable of efficiently simulating, translating, and analyzing high-energy physics (HEP) data generated by the simulated BabyCal electromagnetic calorimeter. By leveraging HEP data simulation software, computer clusters, and cutting-edge machine learning algorithms, such as convolutional neural networks (CNNs) and autoencoders, the system effectively manages a dataset of approximately 10,000 entries.

The identified problems for this work were able to be resolved through various approaches. To address data generation and extraction, automated simulation processes were developed, mitigating the need for extensive expert knowledge in handling GEMC. The process for massive data generation is much more straight forward now, with scripts that use more standard technologies for easier understanding. Consequently, a large volume of data was successfully generated to train the machine learning tool. For data transformation, efficient formatting methods were developed to conduct data delivery to the machine learning tool. Additionally, cutting-edge machine learning algorithms facilitated data analysis for multiple simulations.

The development of The Framework manages to accelerate the research in high-energy physics. Its capabilities already have the potential to aid in investigations that require a lot of analysis from large amounts of data and don’t always have the resources to fully achieve their potential.

The Framework is expected to be used by professionals of the areas of Physics and Computer Science. With this framework at their disposal, researchers can tackle complex investigations reducing the time spent in preparing the data and instead focus on understanding it.

With that, the model is expected to increase it’s capabilities, allowing for more diverse and larger datasets for the study of HEP particles. This way, The Framework can include more machine learning algorithms for a wide variety of tasks.

In conclusion, the main objective of this thesis work was successfully achieved through the development of The Framework, which consists in the integration of a diverse set of tools, that allow for an efficient data workflow and an easy-to-use set of commands. Through this integration, the previously independent components come together to form a cohesive system where data simulation, transformation and analysis are effortlessly streamlined.

5.2 SPECIFIC CONCLUSIONS

The integration of data simulation software, computer clusters, and machine learning algorithms has allowed the system to handle scalable datasets, expediting research and accelerating the analysis of particles and phenomena. The specific objectives of this thesis work were successfully achieved by developing each one and integrating the tools to work as one cohesive system:

Massive amounts of data were generated by leveraging the computational power of the CCT-Val's computer cluster and GEMC, allowing for automated data simulation on the cluster. GEMC offers a wide variety of very precise ways of simulating an infinity of physical scenarios. The ones developed in this work were barely 3. The potential for using GEMC to expand the Framework capabilities from now on is unlimited.

The data was transformed using Gruid and processed to be delivered to various machine learning algorithms. Gruid offers a very specific fit for the BabyCal, it is a very easy-to-use tool that allows for fast data handling and translating. It's modular nature, design and compatibility with state-of-the-art technologies, make it essential for The Framework to work properly.

The data was analyzed with CNNs and autoencoders which were able to process the vast amount of data generated by the automated process. It's capabilities are already fit for processing even higher amounts of data.

The encountered challenge of distinguishing between muons and antimuons within the dataset has shed light on the complexity of particle classification, highlighting the need for larger and more diverse datasets to achieve more accurate distinctions.

To improve the CNN's generalization capabilities, regularization techniques such as dropout (blocking neurons o whole layers) or early stopping can be employed during training. Additionally, data augmentation (generating new synthetic data out of the already existing one) and larger datasets may enhance the model's ability to identify patterns and achieve better separability between the classes.

The success of the autoencoder in reconstructing antimuon data underscores the potential of alternative classification methodologies. To gain deeper insights, exploring the MSE error with other particle types is recommended. Evaluating how the autoencoder behaves with different particle data could tell whether the MSE error rises or remains consistent, providing a better understanding of the autoencoder's performance and its ability to distinguish between various particle types. In conclusion, the MSE plot reveals interesting findings in computational terms, indicating that muons and antimuons exhibit very similar MSE values during reconstruction. Further analysis with additional particle types could perhaps provide valuable information and contribute to a more comprehensive understanding of the autoencoder's capabilities in reconstructing particles.

5.3 RECOMMENDATIONS

The code for the developed automated system, including the CNN and autoencoder implementations for high-energy physics data analysis, can be found on this thesis work's publicly accessible repository [Hebel, 2023]. I have made the code available for the scientific community of the CCTVal, my teachers and myself, to promote and facilitate collaboration within the field. This will allow fellow students and researchers to explore, reproduce, and build upon the work, aiming for expandig The Framework's capabilities and for further advancements in particle physics data analysis with the BabyCal.

5.4 FUTURE WORK

This thesis work had promising results and opens up a lot of potential future work. This includes testing the models with new particles to see if a machine learning tool can manage to classify particles based only on their spatial behaviour, getting more diverse and extensive training data for improving the generalization capabilities of the models, trying new model settings with hyper-parameter tuning, updating the system components to improve compatibility with new technologies, trying different optimizers and collaborating with physicists to ensure coherence in the purpose of the analysis and the goals of the work to come.

Adding control scripts for job execution in the cluster is also an important consideration. It will help to optimize the generating rate of simulations, since The Framework doesn't currently control the loss of jobs due to user limitations when the cluster's queue is overloaded.

Containerizing future versions of The Framework with Singularity offers great potential for enhancing its overall scalability. By encapsulating the framework and its components in a portable and isolated container, users could ensure seamless deployment across different HPC environments. As the framework evolves with better components, this containerized approach could allow for upgrades and ensure compatibility with future needs.

5.5 AUTHORS' FINAL WORDS

In these final words, I express my immense gratitude to my academic mentors and all those who supported me throughout this thesis work journey. Achieving the objective of developing an automated system for HEP data analysis fills me with great pride and satisfaction. The challenges encountered along the way have provided me with invaluable insights and learning experiences. I appreciate the opportunities for growth and collaboration that this thesis work has offered. Looking ahead, I am excited about the potential impact of the automated system on high-energy physics investigations with the BabyCal, and I hope its role in inspiring further advancements in this fascinating field gets to be an interesting one.

REFERENCES

[Aad et al., 2012] Aad, G., Abajyan, T., Abbott, B., Abdallah, J., Abdel Khalek, S., Abdelalim, A., Abdinov, O., Aben, R., Abi, B., Abolins, M., and et al. (ATLAS Collaboration) (2012). Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29.

[Albertsson et al., 2018] Albertsson, K., Altoe, P., Anderson, D., Andrews, M., Araque Espinosa, J. P., Aurisano, A., Basara, L., Bevan, A., Bhimji, W., Bonacorsi, D., et al. (2018). Machine learning in high energy physics community white paper. In *Journal of Physics: Conference Series*, volume 1085, page 022008. IOP Publishing.

[Bailey et al., 1979] Bailey, J., Borer, K., Combley, F., Drumm, H., Eck, C., Farley, F., Field, J., Flegel, W., Hattersley, P., Krienen, F., Lange, F., Lebée, G., McMillan, E., Petrucci, G., Picasso, E., Rúnolfsson, O., von Rüden, W., Williams, R., and Wojcicki, S. (1979). Final report on the cern muon storage ring including the anomalous magnetic moment and the electric dipole moment of the muon, and a direct test of relativistic time dilation. *Nuclear Physics B*, 150:1–75.

[Baldi et al., 2014] Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1).

[Beattie et al., 2018] Beattie, T., Foda, A., Henschel, C., Katsaganis, S., Krueger, S., Lolos, G., Papandreou, Z., Plummer, E., Semenova, I., Semenov, A., Barbosa, F., Chudakov, E., Dalton, M., Lawrence, D., Qiang, Y., Sandoval, N., Smith, E., Stanislav, C., Stevens, J., Taylor, S., Whitlatch, T., Zihlmann, B., Levine, W., McGinley, W., Meyer, C., Staib, M., Anassontzis, E., Kourkoumelis, C., Vasileiadis, G., Voulgaris, G., Brooks, W., Hakobyan, H., Kuleshov, S., Rojas, R., Romero, C., Soto, O., Toro, A., Vega, I., and Shepherd, M. (2018). Construction and performance of the barrel electromagnetic calorimeter for the gluex experiment. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 896:24–42.

[Behroozi, 2014] Behroozi, F. (2014). A Simple Derivation of Time Dilation and Length Contraction in Special Relativity. *The Physics Teacher*, 52(7):410–412.

[Benkel, 2021] Benkel, B. (2021). bleaktwig/gruid-translator: Small tool in python that converts gemc simulations to a time series of matrices. URL: https://github.com/bleaktwig/gruid-translator (visited on 19/11/2021).

[Bhatt, 2018] Bhatt, S. (2018). Reinforcement learning 101. URL: https://www.ibm.com/topics/automation (visited on 08/03/2023).

[CCIN2P3, 2012] CCIN2P3 (2012). Cad - gdml. URL: http://cad-gdml.in2p3.fr (visited on 19/11/2021).

[CCTVal, 2021] CCTVal (2021). Experimental high-energy physics. URL: https://cctval.usm.cl/en/lines-of-investigation/experimental-high-energy-physics/ (visited on 19/11/2021).

[CERN, 2022] CERN (2022). The standard model. URL: https://home.cern/science/physics/standard-model (visited on 02/08/2022).

[CERN, 2023a] CERN (2023a). Lhc | home.cern.ch. URL: home.cern/science/accelerators/large-hadron-collider (visited on 19/11/2021).

[CERN, 2023b] CERN (2023b). Overview | geant4.web.cern.ch. URL: https://geant4.web.cern.ch (visited on 19/11/2021).

[Chatrchyan et al., 2012] Chatrchyan, S., Khachatryan, V., Sirunyan, A., Tumasyan, A., Adam, W., Aguilo, E., Bergauer, T., Dragicevic, M., Erö, J., Fabjan, C., and et al. (CMS Collaboration) (2012). Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1):30–61.

[CODATA, 2020] CODATA (2020). Framework. URL: https://codata.org/rdm-terminology/framework/ (visited on 19/11/2021).

[Dertat, 2017] Dertat, A. (2017). Applied deep learning - part 3: Autoencoders. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 (visited on 06/05/2023).

[Gerber, 2018] Gerber, L. (2018). *Containerization for HPC in the Cloud: Docker Vs Singularity a Comparative Performance Benchmark*. UMEA UNIVERSITY.

[Guest et al., 2018] Guest, D., Cranmer, K., and Whiteson, D. (2018). Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, 68(1):161–181.

[Hebel, 2023] Hebel, D. (2023). dhebel/babycal: A framework for data simulation and analysis of the babycal electromagnetic calorimeter. URL: https://github.com/dhebel/babycal.

[Hosny et al., 2018] Hosny, A., Parmar, C., Quackenbush, J., Schwartz, L. H., and Aerts, H. J. W. L. (2018). Artificial intelligence in radiology. *Nature Reviews Cancer*, 18(8):500–510.

[IBM, 2023] IBM (2023). What is automation? URL: https://www.ibm.com/topics/automation (visited on 08/03/2023).

[JLAB, 2021] JLAB (2021). Thomas jefferson national accelerator facility. URL: https://www.jlab.org/physics/hall-b/clas12 (visited on 19/11/2021).

[Keras Team, 2023] Keras Team, T. (2023). keras/keras/metrics/accuracy_metrics.py at master · keras-team/keras. URL: https://github.com/keras-team/keras/blob/master/keras/metrics/accuracy_metrics.py (visited on 20/06/2023).

[Lecun et al., 2015] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Molina, 2021] Molina, E. (2021). Bcal. URL: https://github.com/emolinac/bcal (visited on 19/11/2021).

[NM State University, 2021] NM State University, N. S. U. (2021). Running program in series vs parallel :: High performance computing. URL: https://docs.sylabs.io/guides/3.5/user-guide/index.html (visited on 20/08/2022).

[Peng et al., 2021] Peng, J., Jury, E., Dönnes, P., and Ciurtin, C. (2021). Machine learning techniques for personalised medicine approaches in immune-mediated chronic inflammatory diseases: Applications and challenges. *Frontiers in Pharmacology*, 12:Page 2.

[Powers, 2020] Powers, D. M. (2020). Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

[Radovic et al., 2018] Radovic, A., Williams, M., Rousseau, D., Kagan, M., Bonacorsi, D., Himmel, A., Aurisano, A., Terao, K., and Wongjirad, T. (2018). Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560(7716):41–48.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

[Singularity Docs, 2019] Singularity Docs, S. D. (2019). User guide - singularity container 3.5 documentation. URL: https://docs.sylabs.io/guides/3.5/user-guide/index.html (visited on 20/08/2022).

[Swapna, 2020] Swapna, K. (2020). Convolutional neural network | deep learning. URL: https://developersbreach.com/convolution-neural-network-deep-learning/ (visited on 12/03/2023).

[Ungaro, 2019] Ungaro, M. (2019). Geant4 monte-carlo. URL: https://gemc.jlab.org/gemc/html/index.html (visited on 19/11/2021).