DIPLOMARBEIT

Data Analysis of the XBox-2 Radiofrequency Cavity at CERN using Machine Learning Techniques

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Lorenz Fischl, BSc

Matrikelnummer 01625248

ausgeführt am Institut für Analysis und Scientific Computing
der Fakultät für Mathematik und Geoinformation

unter der Leitung von Ass. Prof. Dipl.-Ing. Dr.techn. Michael Feischl

Wien, Mai 2022

MASTER THESIS

Data Analysis of the XBox-2 Radiofrequency Cavity at CERN using Machine Learning Techniques

submitted for the degree of

Diplom-Ingenieur

in the master program

Technical Mathematics

submitted by

Lorenz Fischl, BSc

Student ID 01625248

at the Institute of Analysis and Scientific Computing
Faculty of Mathematics and Geoinformation

under the supervision of Ass. Prof. Dipl.-Ing. Dr.techn. Michael Feischl

Vienna, May 2022

1 Abstract

This master's thesis starts with an introduction to particle physics. Thereby, the basic operating principles of high-gradient linear accelerators are explained. One of the main limitations in these devices is the occurrence of breakdowns, which is investigated in the experimental accelerating structure XBox-2 located at CERN. An adaptable framework for data analysis using machine learning is created with the goal of deriving analysis results from raw experimental data. A strong focus lies on its optimized implementation, which is described in detail. The framework is applied to the data of the XBox-2 accelerator with unsupervised and supervised machine learning techniques. A hypothesis for breakdown indicators is derived from the trained models and tested in the lab. However, further testing on accelerating structures is required before the results of the analysis can be validated.

2 Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume. If text passages from sources are used literally, they are marked as such.

I confirm that this work is original and has not been submitted elsewhere for any examination, nor is it currently under consideration for a thesis elsewhere.

Vienna, May 2022

Lorenz Fischl, BSc

3 Acknowlegements

I would like to thank my supervisor Michael Feischl for his great support before, during and after my semester abroad at CERN. This thesis would not be possible without his support for my master thesis in cooperation with CERN.

I would like to thank my colleges at CERN for the work experience in the greatest international collaboration environment I could ever imagine. I am especially grateful to Christoph Obermair who helped and guided me during my whole stay. Milosz Blaszkiewicz was not only a great work college but also became a friend during my stay in Geneva. Lee Millar always helpfully answered my questions on the engineering of particle accelerators.

I would like to thank my parents for their endless support in education financially, emotionally, and by spending time with me. They truly embraced the importance of their children's intellectual development.

I would like to thank my partner who supported me throughout my whole studies and my yet unborn child who gave me additional motivation to work hard on this thesis.

I would like to thank my colleges at my weight lifting sports clubs *Eiche Wien Ottakring* and *Polizei SV* with whom I was able to recover from long hours of studying with physical training.

Contents

4 Introduction

4.1 Overview

The investigation of high energy phenomena in particle accelerators is the essential part of modern particle physics. The research has improved the understanding of the standard model, which is a model of the fundamental building blocks of matter. Larger and more powerful machines for acceleration are needed in order to explain additional phenomena. Additionally, this technology is used in other areas like cancer treatment which benefit from the technological advances.

Data analysis with machine learning has proved to be effective for the analysis of experimental data. In machine maintenance and protection at CERN, however, machine learning is more novel. Often customized solutions are created for each project, which is a tedious and time consuming process. Instead of individual solutions a universal machine learning framework might increase the quality of data analysis and its work flow. Furthermore, new machine learning features are developed regularly. Analysis does not only have to be flexible to adapt to new data but also reproducible to apply new machine learning techniques.

The primary source of information for the theoretical background in high energy physics were the CERN Accelerator School courses, two books on the topic of special relativity by Gunther Helmut and Woodhouse Nicholas [Gun13]; [Woo16] and the book *Particle Physics* by Anwar Kamal [Kam14]. Plans of future accelerators at CERN and the technology of linear accelerators are published by CERN in press statements, the implementation plan and the summary report of the CLIC project [Bur+18]; [Aic+18]. The second part of this thesis focuses on the efficient implementation of the data transformation and manipulation of the machine learning framework used in *Explainable Machine Learning for Breakdown Prediction in High Gradient RF Cavities* by Christoph Obermair [Obe+22]. There, an in-depth analysis of the experimental data, the device under test and the physical explanation of the analysis results is done. Machine learning is currently a very active field of research. Thus the techniques for the design of machine learning models and their evaluations are a combination of suggested techniques and new implementations. The *tensorflow* package by Google was used for neural networks, the *UMAP* Algorithm by McInnes Leland was used for data exploration [MHM18], and the *shapely* values by Lundberg Scott and Lee Su-In [LL17] were used for the sensitivity analysis. The application of different machine learning models on the same data is done in [Obe+22].

This thesis first presents a soft introduction into theoretical physics that is build on a solid axiomatic basis. This enables a theoretical analysis of the relativistic me-

chanics of high energy physics experiments. A comparison of different accelerator and collider designs is done on basis of those results. This thesis further compares linear to circular accelerators and introduces the XBox-2 test stand, a klystron-based high power test facility at CERN which has been established as part of the CLIC (compact linear collider) project at CERN. Subsequently, a framework for data analysis with machine learning is created and applied to XBox-2 test data. It is designed to be reused for the application of machine learning techniques onto other projects. A thorough comparison of data formats is done with an analysis of their data structure and their implementation possibilities for the framework. The required storage space and the read- and write-speeds are compared on example of the XBox-2 data set. The framework comprises data cleaning, calculating machine learning features, training a machine learning model to predict faulty behavior of the accelerating unit, and analyzing its predictions with a sensitivity analysis. Multiple graphical illustrations of structural or functional properties of the framework help to visualize how the data is processed. Additionally, its implementation is presented in the appendix.

4.2 Relativistic Physics

In this section classical mechanics are introduced on the basis of Newton's laws of motion. Together with the postulates of special relativity by Einstein, they form relativistic mechanics. Finally, the Lorentz equation, which describes the electromagnetic force, is introduced.

4.2.1 Classical Mechanics

A frame of reference or observer is referred to as the position in space and time from which objects are observed. The observer is always the center of the Cartesian coordinate system.

In Newton mechanics objects are referred to as bodies of matter that follow a three dimensional path $s : \mathbb{R}^+ \to \mathbb{R}^3$, with velocity $v : \mathbb{R}^+ \to \mathbb{R}^3$ and acceleration $a : \mathbb{R}^+ \to \mathbb{R}^3$. This path depends on the frame of reference, the observer.

Newton's first law of motion defines what an inertial reference frame is. It states:

> *Lex I: Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus a viribus impressis cogitur statum illum mutare.*
> *Law I: An object stays still or will continue to move in a straight line, when the net sum of forces that act on it is zero. [New40]*

According to the second law, which will be presented below, this is only true if the observer is in a frame of reference where no acceleration and no rotation happens. Such an observer is referred to as an inertial frame of reference. This axiom ensures the existence of such a frame of reference.

Newton I Let there be a frame of reference Σ_0. If each object that experiences a net force $F : \mathbb{R}^+ \to \mathbb{R}$ of zero experiences no acceleration $a : \mathbb{R}^+ \to \mathbb{R}^3$, then the frame of reference Σ_0 is called inertial, i.e. $F = 0$ (constant and zero) implies $a = 0$.

The first law of motion is often called the law of inertia.

The second law of motion is the most common one of the Newton's laws. It defines the relation of acceleration and force applied on objects.

> *Lex II: Mutationem motus proportionalem esse vi motrici impressae, et fieri secundum lineam rectam qua vis illa imprimitur.*
> *Law II: The change of motion of an object is directly proportional to the net force that acts on it, the direction of change is equal to the direction of the net force. [New40]*

The change of motion is called acceleration. The factor with which the net force and the acceleration are proportional depends on the units and the object's mass. The product of mass and velocity is called momentum and will help with the formulation of the second law of motion.

Newton II Let the frame of reference be inertial and contain an object of mass $m : \mathbb{R}^+ \to \mathbb{R}^+$. Let the net force that acts on the object be given by $F : \mathbb{R}^+ \to \mathbb{R}$ and the velocity of it be given by the function $v : \mathbb{R}^+ \to \mathbb{R}^3$. Then the momentum $p = mv$ satisfies $F = \frac{dp}{dt}$.

The second law of motion is often shortened to $F = m\frac{dv}{dt} = ma$ if the mass of an object is assumed to be constant. However, we will later need the concept of variable mass or rather relativistic momentum.

The third law of motion is commonly known by the Latin phrase *action et reaction* which means that every impulse of motion comes form a counter impulse within a closed system.

> *Lex III: Actioni contrariam semper et aequalem esse reactionem, sive corporum duorum actiones in se mutuo semper esse aequales et in partes dirigi.*

Law III: Every force acting on an object causes a counter force, which is equally strong and points in the opposite direction. [New40]

The conservation of momentum can be directly derived from this law. It states that the sum of all momentums stays constant. For each force applied on an object there is a counter object on which an equally large force is applied.

Newton III Let Σ_0 be an inertial frame of reference and A and B be two objects in it. Let the force that acts on B by A be called $F_{AB} \in \mathbb{R}$ and the force that acts on A by B be called $F_{BA} \in \mathbb{R}$ then it holds that $F_{AB} = -F_{BA}$.

We will now formulate Newton's third law of motion for n different objects. Outer forces of the i-th object are referred to as F_i. If all F_i are zero then a force that acts on an object i is caused by another object j. For each of those interactions the third law of motion applies so that for all $i, j \in \{1, \dots, n\}$ the force F_{ij} equals its counter force $-F_{ji}$. The sum of all entries of this anti symmetric matrix $[F_{i,j}]_{i,j=1,\dots,n}$ is $\sum_{j=0}^{n} \sum_{i=1}^{n} F_{ij}$ which is zero.

The sum over all particles momenta is called the total momentum $P := \sum_{i=0}^{n} p_i$. Its derivative is the sum of all forces $\frac{d}{dt}P = \sum_{i=1}^{n} \sum_{j=1}^{n} F_{ij} + \sum_{i=1}^{n} F_i$ because of the second law of motion. We assumed the outer forces F_i to be zero and calculated that the sum of inner forces cancel each other out. Consequently, the derivative of the total momentum $\frac{d}{dt}[P]$ is zero[New40, chapter 19].

Newton's laws of motion describe classical mechanics for speeds $v \ll c$. They can be summarized by

$$\begin{aligned}
\mathcal{N}_1: &\quad F = 0 \Longrightarrow \frac{d}{dt}[p] = 0, \\
\mathcal{N}_2: &\quad F = \frac{d}{dt}[p]\frac{d}{dt}[mv], \\
\mathcal{N}_3: &\quad F_{AB} = -F_{BA} \qquad \text{or} \qquad \forall i \; F_i = 0 \Longrightarrow \frac{d}{dt}P = 0.
\end{aligned}$$

4.2.2 Special Relativity

Time is the fundamental concept of the mathematical model of the universe. It divides events into past, present and future. The position of an object is typically given as a function from time to space and with it its velocity and acceleration. In classical mechanics time is defined to be the same everywhere, regardless of the position in space and to pass at a constant rate regardless of the movement in space. In 1905 Albert Einstein published the theory of *special relativity* where

time is defined relative to the observer. The foundation of special relativity is given by the two following postulates:

> *§1 The laws by which the states of physical systems undergo change are not affected, whether these changes of state be referred to the one or the other of two systems of coordinates in uniform translatory motion. §2 Any ray of light moves in the "stationary" system of coordinates with the determined velocity c, whether the ray be emitted by a stationary or by a moving body. Hence velocity* $= \frac{\textit{light path}}{\textit{time interval}}$ *where time interval is to be taken in the sense of the definition in § 1. [Ein+05]*

In the first postulate of special relativity Einstein states that laws of physics remain true for each inertial frame of reference (see Section 4.2.1). We will later observe that mechanics in special relativity follows an adapted version of Newton's laws of motion. The second postulate states that the speed of light $c = 299792458\text{m/s}$ is the same regardless of the motion of the observer relative to the source of light. As a consequence time has to pass at a different rate (dilation) and space has to be contracted for an object that moves relative to the observer.

In order to create a mathematical model for special relativity where the postulates hold true, Einstein uses a different synchronization process for time than that in classical mechanics: Let the observer A send light to an object B at time t_A located ℓ away. When the light is detected at B the time is set to $t_B = t_A + \frac{\ell}{c}$. This synchronization process defines events to be simultaneous when the observer A observes them as simultaneous events. This process is called the Einstein time synchronization process while in the time of classical mechanics it is called the classical time synchronization [Gun13, Chapter 4].

For the notation of tensors, the sums of vectors or matrix vector multiplications the Einstein notation is used. Similar upper and lower indices represent a sum over that index.

Definition 4.1 (Einstein notation)**.** *Let p and q be vectors of length $n+1$ with $\vec{p} = (p_0, p_1, \ldots, p_n) \in \mathbb{R}^{n+1}$ and g be a matrix of size $(n+1)^2$ with according indices. Then the Einstein's summation notation is given by*

$$p_\mu q^\mu := \langle p, q \rangle = \sum_{i=0}^{n} p_i q_i$$

$$g_{\mu\nu} p^\mu q^\nu := \langle \langle g, p \rangle, q \rangle = \sum_{i=0}^{n} \sum_{j=0}^{n} g_{i,j} p_i q_j$$

Note that a vector v in a vector space of dimension $n+1$ can be represented in basis $\{b_0, \ldots, b_n\}$. Thus it holds that $v = v_\mu b^\mu$.

In special relativity, a fourth dimension is added to the three space dimensions in order to keep track of the time which depends on the observers location and speed. Consider an object B located ℓ away from the observer with a time t_B synchronized with the Einstein synchronization process with the observer's time t_A. The time component that is assigned to B is given by $ct_B = ct_A + \ell$. The point in time t_B is stored in the 0-th component as the distance light travels from the beginning of time 0 to t_A plus the distance it travels from A to B.

The time coordinate and the space coordinates are given in length unit m (meters). The time component is usually written as the multiplication of time in s (seconds) and the speed of light c. Note that by storing the time component in meters, the 4-dimensional vectors have equal units in each component. The space of 4-vectors (ct, x, y, z) is called Minkowski space. The metric tensor

$$g_{\mu\nu} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

induces a pseudo Euclidean space M, which is a real vector space with an indefinite inner product. The inner product of the Minkowski space of $a = (ct_a, x_a, y_a, z_a)$ and $b = (ct_b, x_b, y_b, z_b)$ is thus given by $g(a, b) = g_{\mu\nu} a^\mu b^\nu = a_0 b_0 - \sum_{i=1}^{3} a_i b_i = c^2 t_a t_b - x_a x_b - y_a y_b - z_a z_b$ [Gun13, Chapter 33].

According to the second postulate of special relativity the speed of light is the same in every inertial system. From that we deduce that objects that move relative to the observer experience space contraction and time dilatation. This means that time goes by differently for moving objects and lengths appear contracted from the viewpoint of the observer. The change from one reference frame or observer to another is called a transformation. The transformations in this thesis will always be between inertial frames of reference. The mathematical model for relativistic transformations between Minkowski spaces is called Lorentz-Transformation. Transformations in classical mechanics between classical vector spaces are isometric affine linear transformations and are referred to as the Galilei-Transformation. Under the Lorentz-Transformation, however, time and lengths are contracted by a factor that depends on the relative velocity of the first to the second observer. The value of the Lorentz-Transformation factor will be calculated in the following example as a scaled version of the Galilei-Transformation.

Example 4.2. *Let there be two inertial systems Σ and Σ' both located at the origin at $t = 0 = t'$ that move with velocity $v = (v_1, 0, 0)$ relative to each other. For simplicity we will only consider the first space coordinate x while y and z stay zero. The system Σ observes the center of Σ' to be at $x' = x + vt$ and the system*

Σ' *observes the center of the system* Σ *to be at* $x = x' - vt'$. *Transforming from one frame of reference to another with the Galilei-Transformation, which is a good approximation for* $v \ll c$, *yields*

$$\begin{array}{rcl|rcl} \Sigma' & \longrightarrow & \Sigma & \Sigma & \longrightarrow & \Sigma' \\ t & = & t' & t' & = & t \\ x & = & x' + vt' & x' & = & x - vt. \end{array}$$

The second postulate of special relativity states that the speed of light is the same for every inertial system. Thus light that is sent away at the origin at the time $t = 0 = t'$ *moves with the speed of light* c, *regardless of the system. Thus the position of the light photons is given by*

$$\begin{array}{cc} \Sigma & \Sigma' \\ x_p = ct & x'_p = ct' \end{array} \tag{1}$$

When using the Galilei-Transformation we realize that it violates the second postulate of special relativity. The speed of the light photon x'_p *transformed into* Σ *has speed* $c+v$ *which is greater than* c. *This is why for faster speeds* v *we expect a time dilatation by a factor* ℓ *and space contraction by a factor* k. *We will first analyze the space contraction factor* k. *We define the relativistic transformation of space to be*

$$\begin{array}{rcl|rcl} \Sigma' & \longrightarrow & \Sigma & \Sigma & \longrightarrow & \Sigma' \\ x & = & k(x' + vt') & x' & = & k(x - vt). \end{array} \tag{2}$$

We will now transform the position of the light photons from Σ *to* Σ' *with the relativistic transformation (2). First, we resolve the position of the photons in* Σ. *Then the relativistic transformation is applied and finally the position of the photons in* Σ' *is used.*

$$ct = x_p = k(x'_p + vt') = k(ct' + vt') \tag{3}$$

Analogously the transition from Σ' *to* Σ *is given by*

$$ct' = x'_p = k(x_p - vt) = k(ct - vt). \tag{4}$$

We multiply both sides of (3) and (4) and obtain

$$c^2tt' = k^2(c - v)(c + v)tt'.$$

We divide by tt', *simplify the equation to* $c^2 = k^2(c^2 - v^2)$, *and finally solve the equation for* k, *which yields*

$$k = \frac{1}{\sqrt{1 - v^2/c^2}}.$$

We will now analyze the time dilatation. We solve (3) for t and obtain $t = k(1 + \frac{v}{c})t'$. By inserting (1) we get alternative version $t = k(t' + \frac{v}{c^2}x')$. Analogously we obtain $t' = k(t + \frac{v}{c^2}x)$.

The relativistic transformation is often referred to as the Lorentz-Transformation. The factor $(1 - v^2/c^2)^{-\frac{1}{2}}$ often appears in the transformation formulas. It is called the Lorentz factor and abbreviated as $\gamma = (1 - \beta^2)^{-1/2}$ with $\beta = v/c$ [Gun13, Chapter 14].

Definition 4.3 (Lorentz-Transformation)**.** *Let Σ and Σ' be two inertial systems that move relative to each other at the speed $\vec{v} = (v, 0, 0)$. The Lorentz-Transformation from one inertial frame of reference to another is given by*

$$\begin{array}{ll} \Sigma' \longrightarrow \Sigma & \Sigma \longrightarrow \Sigma' \\ t = \frac{t' + \frac{v}{c^2}x'}{\sqrt{1 - v^2/c^2}} = \gamma(t' + \frac{v}{c^2}x') & t' = \frac{t - \frac{v}{c^2}x}{\sqrt{1 - v^2/c^2}} = \gamma(t - \frac{v}{c^2}x) \\ x = \frac{x' + vt'}{\sqrt{1 - v^2/c^2}} = \gamma(x' + vt') & x' = \frac{x - vt}{\sqrt{1 - v^2/c^2}} = \gamma(x - vt). \\ y = y' & y' = y \\ z = z' & z = z' \end{array} \tag{5}$$

A relativistic path in the Minkowski space is given by a function from the observer's time to 4 dimensional vectors. The *proper time* of a relativistic path τ is the time measured when following that path which implies its invariance of observer. For a moving or accelerating object the 4-velocity or 4-acceleration-vector is the first or second derivative of the Mikowski space path. Both have equal units for all four dimensions: m/s (meters per second) or m/s^2 (meters per second squared).

In special relativity the mass of a moving object is often said to depend on its velocity. It can be divided into its rest mass, which is the mass at rest relative to the observer, and its relativistic mass. An object with rest mass m_0 moving at velocity v relative to the observer has the relativistic mass $m = \frac{m_0}{\sqrt{1 - v^2/c^2}} = \gamma m_0$. After the publication of special relativity Albert Einstein discovered that special relativity implies the equivalence of mass and energy. Instead of using the concept of mass that changes with its velocity he later suggested only using the concept of rest mass and relativistic kinetic energy in an object. This is possible due to the mass-energy equivalence [Oku89]. This transforms the widely known formula $E = mc^2$ with relativistic mass m to the formula

$$E = \sqrt{m_0^2 c^4 + p^2 c^2} \tag{6}$$

with rest mass m_0.

The 4-momentum-vector $\vec{p}$ is the product of velocity v and the relativistic mass m. It is given in $\mathrm{kg} \cdot \mathrm{m/s}$ (kilogram times meter per second). The 4-energy-vector is the product of the 4-acceleration times the relativistic mass and is given in $\mathrm{kg} \cdot \mathrm{ms}^{-2}$ (kilogram times meter per second squared).

4.2.3 Relativistic Mechanics

As seen above, time dilatation and space contraction influence the functions of velocity, acceleration, and momentum. Consequently, Newton's laws of motion are also affected by relativistic physics. The chosen formulations, however, remain true if interpreted the right way.

The first and second law of relativistic motion are similar when we consider the relativistic 4-momentum-vector instead of the classic momentum. The first law of motion, $F = 0$ then $\frac{dp}{dt} = 0$, remains true when using the relativistic 4-momentum $p(t) = \gamma(v(t))mv(t)$. The second law of motion, $F = \frac{dp}{dt}$, stays true and can be simplified to

$$\begin{aligned} F = \frac{dp}{dt} &= \frac{d\gamma}{dt}mv + \gamma\frac{dm}{dt}v + \gamma m\frac{dv}{dt} \\ &= \gamma^3\frac{v}{c^2}\cdot\frac{dv}{dt}mv + \gamma\frac{dm}{dt}v + \gamma m\frac{dv}{dt} \\ &= \gamma m\frac{dv}{dt}\left(\gamma^2\frac{v^2}{c^2} + 1\right) + \gamma\frac{dm}{dt}v \\ &= \gamma m\frac{dv}{dt}\frac{2 - \frac{c^2}{v^2}}{1 - \frac{c^2}{v^2}} + \gamma\frac{dm}{dt}. \end{aligned}$$

The original formulation of the third law of motion with two forces F_{AB} and F_{BA} proves to be a complex model in special relativity. It requires the force F_{AB} to act on the object B at the location x_B and the force F_{BA} to act on A at the location x_A simultaneously. Timing the simultaneity for both A and B proves to be complex whereas the alternative and equivalent formulation of the third law of motion is simpler. For a number of n objects it holds that if no outer forces act on the system, then $\frac{d}{dt}P = 0$. The total momentum is given by the sum of all n relativistic 4-momentum[Gun13, Chapter 21].

4.2.4 Electromagnetism

Particle accelerators use electromagnetic force to increase the particles' momentum and keep them on their trajectory. Electromagnetism at the macroscopic level is

described very well by Maxwell's equations. However, in this thesis we will only concentrate on the electric and magnetic fields that act on particles with the so called Lorentz force [Woo16, Chapter 8].

Let there be a particle with charge e in an inertial frame of reference, a magnetic field $\vec{B} : \mathbb{R}^+ \to \mathbb{R}^3$ and an electric field $\vec{E} : \mathbb{R}^+ \to \mathbb{R}^3$. The electric field and the magnetic field act on the particle via the Lorentz force F given by the Lorentz equation

$$F = e\left(\vec{E} + v \times \vec{B}\right)$$

where $\times$ denotes the cross product of v and B. This already shows that magnetic fields can only deviate a particles flight path orthogonal to its momentum. Electric fields, on the other hand, can apply any force. Therefore, in particle accelerators magnetic fields are used only for particle path deflection while electric fields are used for acceleration.

4.3 Standard Model

The *standard model* of elementary particles contains two main groups: *fermions*, building blocks of matter, and *bosons*, the force carriers that are responsible for the interaction between particles. All known matter in the universe is built from these fundamental particles.

A visualization of the standard model is displayed in Figure 1. Each square with rounded edges represents a fundamental particle. We refer to the legend in the top right corner. In each square the name and its abbreviation are written in the right bottom corner and on the right hand side. The mass-energy (see mass-energy equivalence in Section 4.2.2) at rest is written in the top left corner and the charge of the particle is located on the left hand side. The fermions are located in the three columns on the left. The first column contains the first generation of fermions, the second column the second, and the third column the third generation. Generation has nothing to do with the decay of these particles. It characterizes similarities between same generation fermions. The fermions are further subdivided into quarks, that are located in the top two rows in shades of green, and leptons located in the bottom two rows in shades of blue. The intensity of the color correlates with the stability of the fundamental particles. The bosons and the explanation field are displayed on the right hand side of the figure. The bosons are depicted in shades from red to yellow depending on their strength. The particles are further grouped by their interaction with the fundamental forces. Again, the intensity of the color signals the strength of the force. Additionally, the photon and the Z boson point to the Higgs-Boson as it interacts with both forces via the electroweak interaction,

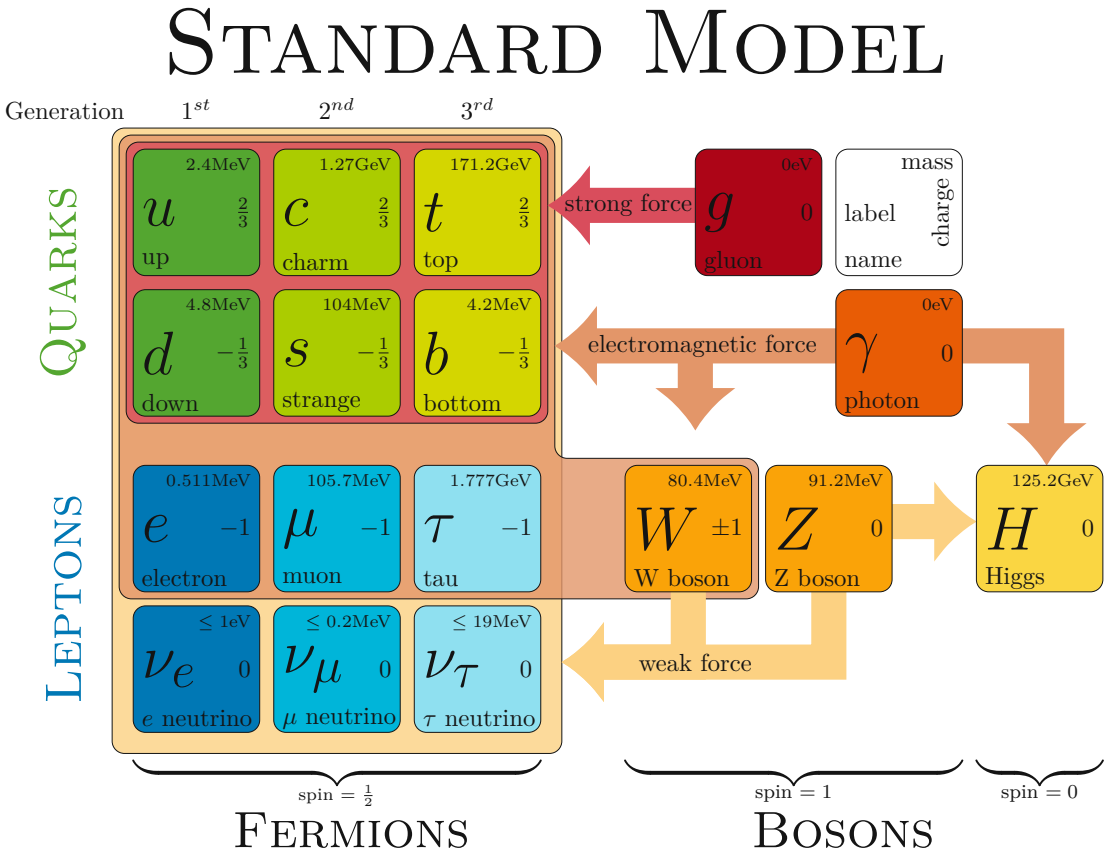


Figure 1: The standard model of particle physics

which is still and active field of research.

Fermions can be subdivided into three generations of particles. Matter that humans interact with contains only the first generation, which is most stable. It contains the electron, the up quark, and the down quark. Two down quarks and one up quark form a neutron and two up quarks and one down quark form a proton. These subatomic particles are called hadrons. Atoms consist of protons, neutrons and electrons. A higher generation fermion has more mass and is less stable than its smaller generation version. The up quark has the charm and the top quark as higher generation ancestors while the down quark has the strange and the bottom quark. The particles in these two triples are called quarks, the other two are called leptons. The first lepton triple is the electron with the muon and tau as ancestors, while the second triple consists of their neutrino pairs.

Interaction with fermions is possible via the four fundamental forces: weak force, strong force, electromagnetic force and gravity. The transmission of force on parti-

cles is done by bosons, the force carriers. The strong force has a very limited range and only acts on quarks. It only acts on a subatomic level where it plays a big role in keeping the atomic nuclei together. The strong force is carried by the gluon. The weak force also solely acts on the subatomic level and is, as the name suggests, much weaker than the strong force. The weak force is mediated by the W and Z bosons. The gravitational force is the weakest by several orders of magnitude and only acts on particles with mass. In high energy physics it is neglected because of its weakness, even though it has an unlimited range. It is speculated that a graviton, a force carrying particle for the gravitational force, exists. However, it has never been detected. The electromagnetic force is the second strongest force and has infinite range and therefore plays a big role in particle accelerators. Only a charged particle, the most common one being the electron, interacts with the electromagnetic force.

All fermions have mass, which makes them interact with the gravitational force. The W, Z and the Higgs-Boson have mass while the gluon and the photon do not. The origin of mass of the W and Z bosons was unsolved until the discovery of the Higgs mechanism and the experimental proof of the existence of the Higgs-Boson.

Not all particles have an electric charge. Each of the up quark ancestor triples has a charge of $\frac{2}{3}$ and each of the the down quark triples a charge of $-\frac{1}{3}$. Thus protons have a charge of 1 and neutrons a charge of 0. Each of the electron ancestor triples has a charge of -1 while their neutrino pairs have no electric charge.

Quarks come in one of three color charges: red, blue, and green. The gluon, the particle responsible for the strong force that keeps the groups of particles together, can only be exchanged if the colors of quarks are different. Protons and neutrons need one of each color quark in order to be stable. Leptons and bosons have no color charge.

The spin of a particle defines its angular momentum. All fermions have a spin of $\frac{1}{2}$ and bosons a spin of 1 except for the Higgs-Boson that has no spin.

High energy physics explores the boundaries of the standard model and seeks to verify underlying principles. Understanding the origin of the interaction between particles and the electromagnetic force is essential for the analysis of particle accelerators, where experiments in particle physics are conducted.

4.4 Particle Accelerators Design

The exploration of the smallest known particles and how they interact with each other is investigated in high energy physics. The *standard model* describes the fundamental building blocks of matter and the interacting forces. An overview can

be found in Section 4.3. Until further discovery those particles are assumed to have no inner structure, they are viewed as being point like. The interaction between particles is described by three major forces: the strong and the weak force and electromagnetism. At high energies this theoretical construct can be brought to its boundaries and potentially reveal the need for new additions. Moreover, some of these particles decay in a very short time (e.g. Higgs-Boson has an average lifetime of the order of 10^{-22}s). In order to observe them and to prove their existence in a controlled lab environment one needs to create them. Such creation of matter is described via Einstein's famous identity $E = m_{rel}c^2$, which relates energy and relativistic mass. Hence, high energy enables the creation of high-mass particles. Transforming energy to mass is done by collision of particles. The collision releases energy from the momentum of the collision particles which creates new mass. A Higgs-Boson weighs approximately $\sim 125,38$ GeV [Sir+20] which translates to the rest mass energy of 134 protons. This means that the energy gained by the momentum has to be several times higher than the rest mass of the particles that carry it [Tie+21]. The total mass energy E of a particle is given by formula (6) of Section 4.2.2

$$E^2 = \widetilde{m}^2c^4 + \vec{\widetilde{p}}^{\,2}c^2.$$

In this formula $\widetilde{m} \in \mathbb{R}^+$ is the rest mass in kg, $\vec{\widetilde{p}} \in \mathbb{R}^3$ is the momentum in kg m/s, and $\vec{\widetilde{p}}^{\,2} = \vec{\widetilde{p}} \cdot \vec{\widetilde{p}}$ denotes its Euclidean scalar product. Mass energy equivalence can be used to simplify Einstein's equation. Mass and momentum can be given in the natural unit eV which needs no translation factor c^2 and c^4. Thus the above can be rewritten with mass m and momentum $\vec{p}$ given in eV

$$E = \sqrt{m^2 + \vec{p}^{\,2}}. \tag{7}$$

As pointed out in Section 4.2.3 the relativistic momentum of a particle is given by

$$\vec{p} = \frac{mv}{\sqrt{1 - v^2/c^2}} = \frac{mv}{\sqrt{1 - \beta^2}} = \gamma mv,$$

where $\beta = v^2/c^2$ and $\gamma = (1 - \beta^2)^{-1/2}$. We can read from equation (7) that accelerating a particle increases its momentum and thus increases its energy. On impact the momentum of a particle is changed. Due to the law of conservation of mass-energy, a large amount of energy obtained by the momentum is transferred to mass in form of new particles. This creation process can be observed in particle accelerators.

The 4-momentum vector $\boldsymbol{p}$ is given by its energy and space components $(E, \vec{p}) = (E, p_x, p_y, p_z) \in \mathbb{R}^4$. For a given particle with the 4-momentum-vector $\boldsymbol{p} \in M$ we can show with (7) that the inner product g of $\boldsymbol{p}$

$$g(\boldsymbol{p}, \boldsymbol{p}) = E^2 - \vec{p}^{\,2} = m^2 + \vec{p}^{\,2} - \vec{p}^{\,2} = m^2 \tag{8}$$

is the reset mass squared which is independent of speed. From the independence of speed, which is also called relativistic invariance, we can deduce the invariance under the Lorentz transformation [Per00, Section 1];[LP96, Section 8].

Particle creation requires a lot of energy that is extremely concentrated in time and space in order to be converted to mass. Such high concentrated energies are achieved by acceleration of particles. The stored energy is liberated on collision of an incident particle with a target particle. If the momentum relative to each other is high enough, the mass-energy carried by the particles is released and new matter might be created [Kam14, Chapter 2].

4.4.1 Fixed vs. Dynamic Target Particle Accelerator

In this section two basic particle collider designs will be compared: two particles colliding where one is fixed in the lab system (one of them has momentum zero) and two particles colliding head on (sum of both momenta is zero).

Let p_I (p_T) be a 4-momentum-vector of an incident (target) particle with rest mass m_I (m_T), momentum $\boldsymbol{p}_I$ ($\boldsymbol{p}_T$), and energy E_I (E_T). Particle creation happens in the system of zero momentum Σ_Z due to the conservation of momentum (third law of motion in Section 4.2.3). It is called zero momentum system because the momenta (incident and target particle) cancel each other out ($\overrightarrow{p_{\Sigma_Z}} = \mathbf{0}$). After particle creation the total momentum stays the same. Σ_Z is also known as the center of mass system.

The lab system Σ_L does not necessarily coincide with Σ_Z. The maximal energy usable for particle creation is the total energy of the system Σ_Z, which is given by $E_{\Sigma_Z}^2 = E_{\Sigma_Z}^2 - \overrightarrow{p_{\Sigma_Z}}^2 = g(\boldsymbol{p}_{\Sigma_Z}, \boldsymbol{p}_{\Sigma_Z})$. As pointed out in equation (8) $g(\boldsymbol{p}_{\Sigma_Z}, \boldsymbol{p}_{\Sigma_Z})$ is relativistic invariant. Consequently, it is the available energy for particle creation independent of its observer. We can simplify the total energy squared to

$$\begin{aligned} E_{\Sigma_Z}^2 = g(\vec{p}_{\Sigma_Z}, \vec{p}_{\Sigma_Z}) &= \overbrace{g(\vec{p}_{\Sigma_L}, \vec{p}_{\Sigma_L})}^{\text{lab system}} = (E_I + E_T)^2 - (p_I + p_T)^2 \\ &= E_I^2 + 2E_I E_T + E_T^2 - \vec{p}_I^{\,2} - 2\vec{p}_I \cdot \vec{p}_T - \vec{p}_T^{\,2} \\ &\overset{(7)}{=} m_I^2 + \vec{p}_I^{\,2} + 2E_I E_T + m_T^2 + \vec{p}_T^{\,2} - \vec{p}_I^{\,2} - 2\vec{p}_I \cdot \vec{p}_T - \vec{p}_T^{\,2} \\ &= m_I^2 + m_T^2 + 2(E_I E_T - \vec{p}_I \cdot \vec{p}_T). \end{aligned} \tag{9}$$

We can compare different arrangements of incident and target particles with the total energy in Σ_Z which in turn allows us to compare different particle collider designs.

There are two particle arrangements which will be compared below: A target particle at rest relative to the lab system and a target particle that moves head on

to the incident particle. We consider both particles to have the same rest mass m and to only be able to move along the x-axis.

In the first model we consider a target particle at rest in the lab, which means that its 4-momentum $\boldsymbol{p}_T$ is zero in each component relative to the lab system. The incident particle, however, moves along the x-axis at speed v. With equation (9) the total energy observed by the zero momentum system Σ_Z is given by

$$\begin{aligned} E_{\Sigma_Z}^2 &= m_I^2 + m_T^2 + 2(E_I E_T - \vec{p}_I \cdot \vec{p}_T) \\ &= 2m^2 + 2E_I m \end{aligned}$$

Accelerators are typically designed to be operated with a chosen particle species. Thus m is considered to be constant and given in natural units eV (electron volt). High energy physics is the research of energies $E^2 = m^2 + \vec{p}^2$ that are significantly higher than that obtained by the rest mass m. We will describe the asymptotic behavior of the total energy in Σ_Z with the Landau notation with respect to the energy of the incident particle and the asymptotic behavior of the energy of the incident particle with the Landau notation with respect to its momentum.

$$E_{\Sigma_Z} = \sqrt{2m^2 + 2E_I m} = \mathcal{O}\left(\sqrt{E_I}\right) \overset{(7)}{=} \mathcal{O}\left(\sqrt{\|\vec{p}\|_2}\right) \tag{10}$$

Here $\|\vec{p}\|_2$ denotes the euclidean norm $\sqrt{\vec{p}^2}$.

In contrast, consider a target particle on a trajectory head on with the incident particle, i.e. with the momentum $\vec{p} = -\vec{p}_T = \vec{p}_I$. Similar to above, we consider the rest masses to be equal, i.e. $m = m_I = m_T$. Both particles are moving with the same velocity relative to the lab system, which means that they have to have the same relativistic invariant mass energy. This can also be observed by applying equation (7) which yields $E = E_I = E_T$. The total energy in Σ_Z from equation (9) is given by

$$E_{\Sigma_Z}^2 = m_I^2 + m_T^2 + 2(E_I E_T - \vec{p}_I \cdot \vec{p}_T) = 2m^2 + 2(E^2 + \vec{p}^2).$$

We can combine the particle's mass and momentum to form another energy term $2E^2 = 2m^2 + \vec{p}^2$ and obtain

$$E_{\Sigma_Z} = 2E = \mathcal{O}(E) \overset{(7)}{=} \mathcal{O}\big(\|\vec{p}\|_2\big). \tag{11}$$

Equation (10) and equation (11) show that the accelerator design has a huge impact on the required momentum to obtain a target value E_{Σ_Z}. In an accelerator with a fixed target, the energy E_{Σ_Z} increases only with the square root of the energy of the incident particle. In contrast, in an accelerator with a dynamic incident and

target particle the available mass energy E_{Σ_Z} scales linearly with the mass energy of both particles. An accelerator with a static target (10) is thus clearly inferior to one with a dynamic target (11) in terms of the maximum required momentum per particle. We can see that even with a clever accelerator design the mass energy obtained by the momentum has to exceed mass by several orders of magnitude. In some cases, however, being able to fixate the target is necessary. Research in material studies, medical applications or isotope production all require the target to be stationary to the lab system.

We now calculate the theoretical minimal required mass energy per particle to produce a Higgs-Boson for both collider designs with Equation 9. As pointed out, a Higgs-Boson is 125.38GeV. Therefore the minimal required energy in the zero momentum system Σ_Z is also 125.38GeV. In a particle accelerator that accelerates protons, this would require a single proton to be accelerated to an energy level of at least 8.363 TeV and hit a fixed target. When accelerating both incident and target proton only an energy level of 62.69MeV is required for each proton before the head on collision. This result is a theoretical lower bound on the required mass energy per particle. Much more energy is needed in order to detect the production of a Higgs-Boson in statistically relevant quantities. However, it shows that a particle accelerator with particles moving in opposite direction is favorable to ones with a fixed target. Consequently all modern accelerators for high energy physics use particles colliding head on [Kam14, Chapter 2][Per00, Section 1].

4.4.2 Linear vs. Circular Particle Accelerator

The idea of using time-dependent electric fields for particle acceleration was introduced by Gustav Ising in 1924 in the paper *Prinzip einer Methode zur Herstellung von Kanalstrahlen hoher Voltzahl* (translation: Principle of a method for the production channeled beams with high voltage)[Isi24]. The first linear accelerator was realized by Rolf Widerøe in 1927 and his findings were published in 1928 in the paper *Über ein neues Prinzip zur Herstellung hoher Spannungen* (translation: On a new principle for the production of high voltages) [Wid28]. Particle accelerators have evolved significantly since then. As pointed out in Section 4.2.4, the particles' energy is increased by electromagnetic fields in all accelerators. In DC (direct current) machines acceleration happens continuously, while AC (alternating current) machines accelerate particles in bunches. In this thesis we will only analyze AC accelerators as they are able to work with higher voltages. A series of accelerated bunches is called a beam. In modern accelerators particles can be accelerated to ultra relativistic speeds, $v \approx c$ with a Lorentz factor $\gamma \gg 1$. These accelerators can be divided into two main groups, circular and linear accelerators. The synchrotron is the modern representative of cyclic accelerators where the beam is held

on a circular flight path by strong magnetic fields. This makes particles pass repeatedly though the same instruments for acceleration, positioning, analysis, and finally collision. In linear accelerators many high energy accelerating units are placed consecutively. Each beam, however, is only used once for collision [Kam14, Chapter 2].

Assume particles in a synchrotron on a perfect circular path s, velocity v, and acceleration a with radius r and angular velocity ν:

$$\begin{aligned} s &: \mathbb{R}^+ \to \mathbb{R}^3 : t \mapsto r\left(\sin\left(\frac{\nu}{r}t\right), \cos\left(\frac{\nu}{r}t\right), 0\right), \\ v &: \mathbb{R}^+ \to \mathbb{R}^3 : t \mapsto \nu\left(-\cos\left(\frac{\nu}{r}t\right), \sin\left(\frac{\nu}{r}t\right), 0\right), \\ a &: \mathbb{R}^+ \to \mathbb{R}^3 : t \mapsto \frac{\nu^2}{r}\left(-\sin\left(\frac{\nu}{r}t\right), -\cos\left(\frac{\nu}{r}t\right), 0\right). \end{aligned}$$

According to the second law of relativistic motion, a force is required to change the particles' flight path from a straight line to a circular path. This force is called the relativistic centripetal force and is given by

$$F = \frac{dp}{dt} = \frac{dm\gamma v}{dt} = \gamma m a(t) \tag{12}$$

where m denotes the rest mass.

We will now calculate the required magnetic field for the particles' flight path deviation. We model the electric field $E : \mathbb{R}^+ \to \mathbb{R}^3$ to be zero and the magnetic field to be given by $B : \mathbb{R}^+ \to \mathbb{R}^3 : t \mapsto (0, 0, b)$. Consequently, the Lorentz force from Section 4.2.4 acting on the particle is given by $F = ev \times B$.

In order to find the required magnetic field strength for particles path deviation we set the needed deviating force (12) equal to the Lorentz force.

$$\gamma m a = ev \times B = eb\nu\left(\sin\left(\frac{\nu}{r}t\right), \cos\left(\frac{\nu}{r}t\right), 0\right) = -eb\frac{r}{\nu}a$$

We can now calculate the magnetic field strength in the third direction by

$$b = -\frac{mv\gamma}{er}. \tag{13}$$

Both linear and circular particle accelerators are built in particle physics laboratories. Each design has its advantages and disadvantages which will be pointed out below by analyzing different accelerators at CERN.

5 Particle Accelerators at CERN

CERN is an international organization that specializes in the research of high energy physics. The dominant theory that explains physics of sub atomic particles is the standard model (see Section 4.3). The W and Z boson were predicted by studying the weak interaction or weak force of the standard model. Their existence was proven at the SPS (Super Proton Synchrotron) at CERN. These discoveries have improved the understanding of the weak force and thus the interaction of quarks and leptons on a sub atomic level. The findings were awarded with the Nobel Prize in Physics in 1984 [Nob84]. Furthermore, the Higgs-Boson was predicted by use of the standard model and its existence was also validated at CERN. The proof of existence enhanced the understanding of the origin of mass of elementary particles and was awarded with the Nobel Prize in Physics in 2013 [Nob13].

5.1 The Large Hadron Collider

The modern accelerators at CERN that can produce the highest energy collisions are *dynamic target particle accelerators*. The biggest ones are synchrotrons that are of circular type. The particles are held on their trajectory by strong magnetic fields. At the interaction points large detectors are placed in order to monitor the particle creation of collisions. The largest particle accelerator at CERN is called the LHC (Large Hadron Collider). Figure 2 shows a map of the area where the LHC and SPS (Super Proton Synchrotron) tunnels are located.

The LHC has a circumference of 26.659km. At the LHC some sections of the accelerator are straight. For example, at the detectors particles are on a straight path. However, we assume that the accelerator describes a perfect circle to calculate the turning radius which is given by $\frac{26659}{2\pi} = 4242.912\text{m}$.

The hadrons accelerated for the highest energy experiment in the LHC are protons. A proton consists of two up quarks with a charge of $\frac{2}{3}e$ and one down quark with a charge of $-\frac{1}{3}e$ (see Section 4.3). Thus the total charge of the proton is $1e$ which is equal to $1.602e - 19\text{C}$ and has a rest mass m of $938.272\text{eV} = 1.673e - 27\text{kg}$ [Tie+21].

The highest energy proton-proton collisions in the LHC release energies of 13TeV [22]. With the help of the mass energy momentum formula in equation (7) in Section 4.4.2 we can compute the composition of the total energy. A proton that has a total energy of 6.5TeV has 0.0144% of its energy from its rest mass and 99.9856% of its energy from its momentum. We solve equation (7) for v in order

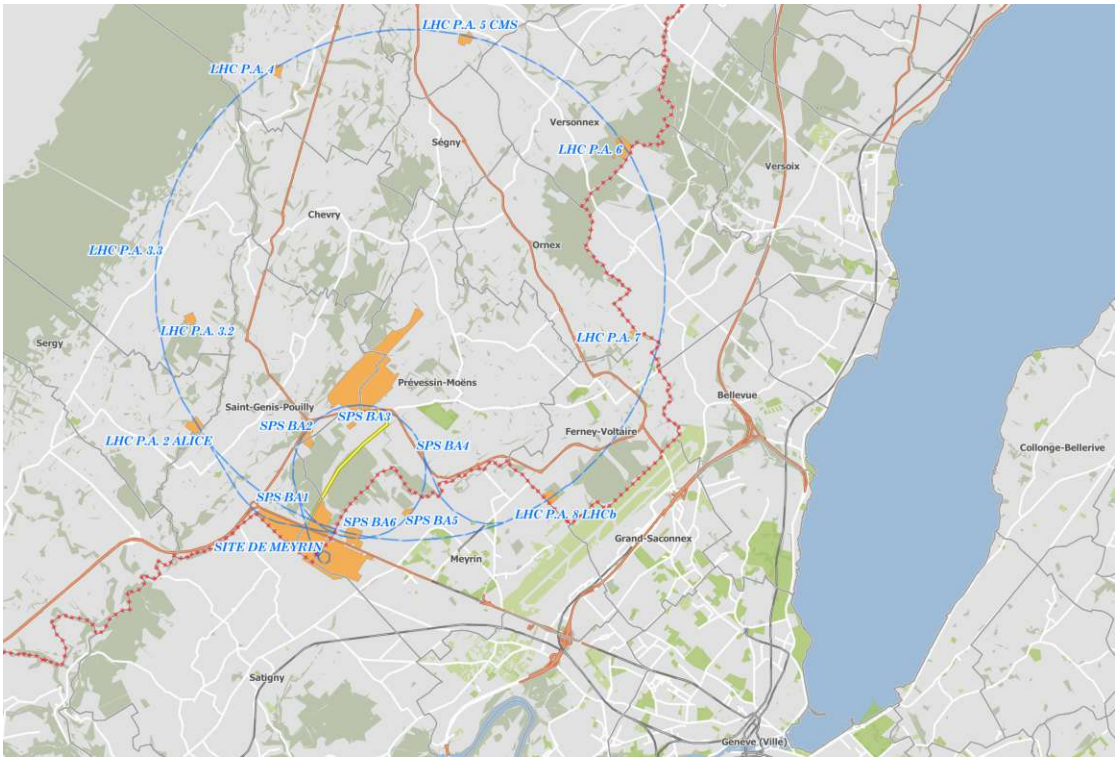


Figure 2: Map of the LHC (large circle) and SPS (smaller circle) at the French-Swiss border [SCE22]

to calculate the speed at which the proton is traveling.

$$v = \sqrt{1 - \frac{m^2}{E^2}} = 299792454.877\text{m/s}$$

The speed v is 99.99999896% of the speed of light, which is just 3.123m/s shy of c. The corresponding Lorentz factor is $\gamma = (1 - v^2/c^2)^{-1} = 6927.627$. Particles that move at a speed of almost c are called *hyper relativistic*.

The magnetic field strength needed for a proton with 6.5TeV to be kept on a circular path in the LHC is given by equation 13 which is

$$b = -\frac{mv\gamma}{er} = -5.112\text{T} = -5.112\frac{\text{m kg}}{\text{C s}^2}.$$

The magnetic field strength b is given in T (Tesla) and SI-units.

5.2 Future Accelerators

Creating heavier particles proves to be a challenge. The deviation of hyper relativistic particles with magnetic fields emits synchrotron radiation which is given by

$$P_\gamma = \frac{\gamma^4 q^2 a^2 (B \times v)^2}{2\pi\varepsilon_0 c^3} = \mathcal{O}(\gamma^4 (v \times \vec{B})^2)$$

where γ is the Lorentz factor, $B \times v$ is the deviating force by the magnetic field, q is the charge, c is the speed of light, and ε_0 is the constant vacuum permittivity. For hyper relativistic particles the gamma factor scales quadratic with the particles' energy. This energy dissipation limits the performance of an accelerator as the increase of particle energy by acceleration has to exceed the loss through radiation. Additionally, the magnetic field strength is already at its limit in the LHC. Currently the LHC uses superconducting niobium-titanium alloy magnets that are already operating at their maximal safe power. Further improvements on the magnets and the application of high temperature superconductors is part of ongoing research. Alternatively a larger turning radius can be used to decrease needed steering force $B \times v$ and thus the energy loss though radiation. An increase of the turning radius r is considered within the FCC (Future Circular Collider) project, which would require a new larger tunnel with $\sim$ 100km circumstance. Both research on more powerful magnets and a research and development study on the FCC are two main bullet points in the ESPP (European Strategy for Particle Physics)[Col+20, Section 3].

Circular accelerators make use of the same accelerating units multiple times at the cost of time and the need of strong magnetic fields which are fragile and expensive. During collision only a fraction of the incident particles collide with a target particle. In the synchrotron the rest of the particles can be sent to collision again, which saves the effort of accelerating the particles again at the cost of the synchrotron radiation. Reusing accelerated particles increases the accelerators performance which is measured in luminosity

$$L = \frac{N_b \; N_T \; N_I \; f}{A}$$

where N_b is the number of bunches used for collision with a frequency f and N_T and N_I are the number of particles per target and incident bunch. Reusing a beam with multiple bunches N_b for collision with a frequency f gives a high luminosity.

In contrast, linear accelerators can accelerate particles to the target energy in a single pass. This design circumvents the limitations associated with path bending in circular machines. It, however, is limited in its luminosity as the frequency of collision f is one. The CLIC (Compact Linear Collider) project at CERN aims

to build such a machine. The plan is to build an increasingly large linear collider that accelerates protons to energies of first 380GeV, then 1.5TeV, and finally 3TeV. This increase in size is possible by concatenation of accelerators [Bur+18, Section 2].

6 X-Band Test Facilities at CERN

6.1 Radio Frequency Cavities

In both circular and linear accelerators, specially designed metallic chambers known as RF (radio frequency) cavities are used for particle acceleration. These establish longitudinal electric field components that accelerate particles. This process is described by the Lorentz equation in Section 4.2.4.

In DC accelerating machines the maximal voltage is limited before electrical arcs are produced. Alternatively periodical switching of voltages reduces the probability of the formation of arcs. As particles gain energy, their velocity increases and thus they require an increased rate polarity switching. Limiting the radiation of electrical waves produced by the fast switching is done by enclosing the structure in a metal chamber. An RF cavity consists of such chambers that are connected with holes where particles and the RF wave travels from one chamber to the next.

In cavities, an oscillatory electromagnetic field (radio frequent) is established and used to impart energy on traversing particles. The cavity design and operating frequency are largely determined by the application, however in the case of CLIC, the use of multi-cell traveling wave linear accelerator cavities is proposed. A diagram showing the principle of operation is shown in Figure 3. When particles enter the first chamber through the first cavity, they experience an electric field between the first and second disk that accelerates them to fly through the second cell's cavity. Due to the oscillation of the electromagnetic wave, the electric field changes polarity right when the particles enter the second chamber. Thus the particles are accelerated throughout every chamber surfing on waves of accelerating electric fields [12].

In Figure 3 a diagram of a multi-cell cavity is displayed. Red arrows mark electric fields accelerating the particles in the direction of their flight path and the blue ones mark the electric fields that would decelerate them [12]. Figure 4 shows a photo of a cavity disk on the hand left and on the right hand side a prototype of the CLIC baseline design that consists of two TD26CC R05 structures.

6.2 The XBox-2 Test Stand at CERN

The XBox-2 test stand is an X-band radio frequency test facility that is used for the development and improvement of the high gradient accelerating units that will be used in the CLIC located at CERN. X-band radio frequency refers to the IEEE (Institute of Electrical and Electronics Engineers) specification of the frequency of

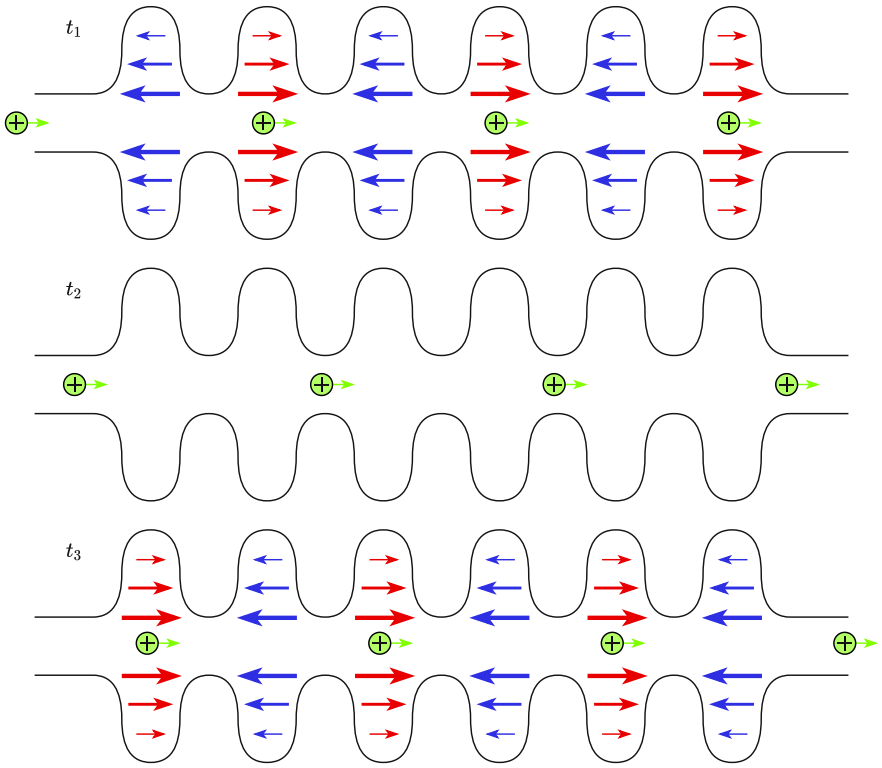


Figure 3: diagram of a multi-cell cavity accelerating at consecutive times t_1, t_2, and t_3 [VM]

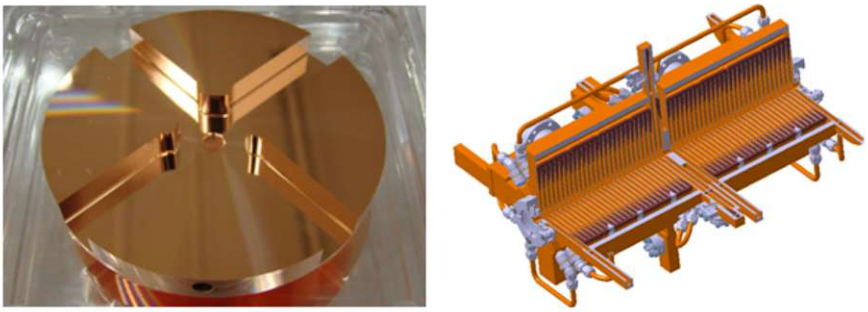


Figure 4: Photo of a single cavity disk that forms a cell (left) and a model of the CLIC prototype structure (right) [VM, Figure 5.17]

the electromagnetic spectrum in the range of $8-13$GHz. The XBox-2 accelerating structure uses 12GHz electromagnetic waves. A picture of one of the XBox-2 accelerating structures is displayed in Figure 5.

Figure 5: The Xbox-2 test stand at CERN

Principal of operation: First, a low level RF signal of a few milliwatt is sent to a *solid state amplifier* (SSA) which amplifies it to a few hundred Watt. The SSA drives a *klystron*, a device that further amplifies the RF pulse to $\sim$ 20MW. Figure 6 shows data measurements of two different pulses as a 2µs time series with 3200 samples. The power amplitude in Figure 6 (a) is measured after the amplification of the klystron and shows a 1.125µs (1800 samples) long pulse. A specially designed device known as *pulse compressor* is filled with RF energy, then discharged to provide a short, high-power RF pulse which is directed to the RF structures or device under test. Figure 6 (b) shows a 0.125µs (200 samples) short pulse of 45MW.

The main limiting factor of high gradient and power RF cavities are *breakdowns*, the occurrence of electrical vacuum arcs. Small deformations in the surface of the device lead to local electric field enhancements which can ultimately lead to the formation of plasma. Figure 6 compares a selections of signals measured during breakdown and healthy events. During the arc formation, material of the surface is melted and thus the vacuum is disturbed, which inhibits particle acceleration and can even damage the RF cavities over time. The plasma formation disturbs the electric fields which reflects the input RF power. The reflected wave is visible in the breakdown pulse of Figure 6 (c) and the lack RF power after the structure in a breakdown pulse can be seen in Figure 6 (d). Charged particles that disturb

the vacuum are accelerated by the electromagnetic waves. These particles can be measured as spikes of dark current at the ends of the RF cavity in both directions called upstream and down stream. Figure 6 (e) and (f) show the dark current that is measured in breakdown events.

Investigating the occurrence of arcs is one of the primary objectives of the CLIC test program, and so the XBox-2 test stand automatically monitors the occurrence of arcs during operation. In the CLIC, where thousands of high gradient RF cavities will be used consecutively, the number of breakdowns per unit and thus the number of shutdowns of the whole structure has to be kept to a minimum. To meet that target luminosity the CLIC project has a target breakdown rate of less than $3 \cdot 10^{-7}$ breakdowns per meter per beam pulse [Bur+18, Section 3].

Breakdowns generally occur in groups as opposed to isolated events. It is assumed that the present deformations and the decreased quality of the vacuum increase the chance of breakdowns [Wue+17]. The first breakdown is called *primary breakdown* and the following ones *follow up breakdowns*. So far, no clear indicator has been found to predict when primary breakdowns happen. The occurrence is assumed to be stochastic. Finding evidence of practical precursors for follow up breakdowns is a topic of current research [Obe+22].

6.3 XBox-2 Dataset

The XBox-2 data discussed in this thesis was recorded from 2018-03-26 to 2018-09-25. Preliminary conditioning process in which the input power is gradually increased over time took place, meaning that the machine parameters were changed regularly. During that time the machine's parameters were adjusted for stable operation, this process is called *conditioning*. After that nine segments, that spanned from a week to a month, were conducted where no settings were changed. These segments are called runs. Generally, the conditions are fixed when operating with a beam, and so breakdowns which occur during such runs are of interest in an operational context. Therefore, only breakdown data during that time will be analyzed.

In the case of the XBox-2 accelerating unit from Section 6.2 the recorded data is given in a native NI (National Instruments) data format called `tdms` which will be explained in detail below. All sensor data of one day is stored in two `tdms` files, the so called event and trend data. The event data contains high resolution data of single events, pulses of RF power that measure the power amplitude of the RF waves. Whereas the trend data contains low resolution data of slowly changing environment variables like the temperature of the structure or pressure in the wave guides.

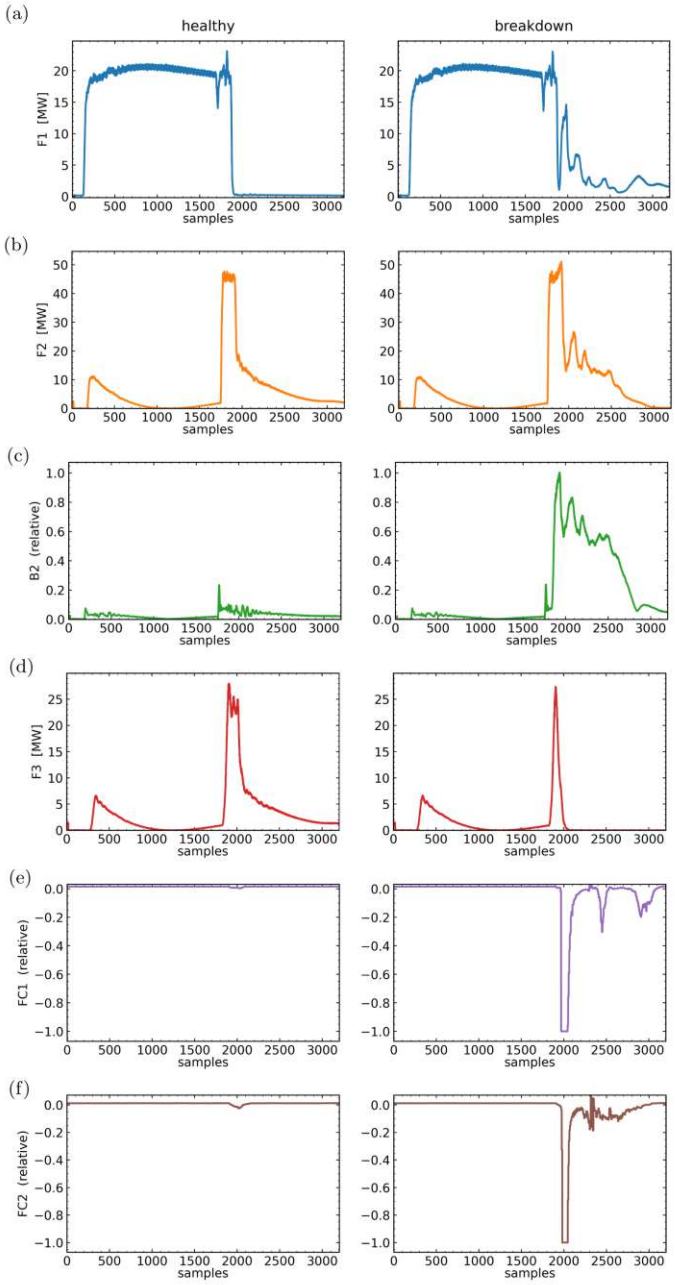


Figure 6: Example signals of healthy events (left) and breakdown events (right). The displayed signals are: the forward traveling wave (F1, F2 and F3) measured before (b) and after (d) the device under test and after the klystron (a), the power amplitude of the reflected wave (B2) traveling backwards(c), and the Faraday cup signals (FC1 and FC2) in both directions upstream (e) and downstream (f). The signals (a), (b) and (d) are given in MW while the signals (c), (e) and (f) are given relative to their maximal values. [Obe+22, FIG. 3]

6.3.1 Tdms files

The tdms (Technical Data Management Streaming) is a file format developed by the company NI that can store technical data. It has been developed to save well documented measurement data produced by lab software like *LabVIEW*. It is a root tree graph based file format with a depth of two. A root tree is an oriented graph in which any two vertices are connected by exactly one path consisting of a series of connected edges. The root vertex is a distinct vertex that is called file in this file format while the children of the file are called groups. The children of the group vertices are channels. All vertices - files, groups, and channels - have additional information stored in dictionaries called properties. The channels are the leaves of the tree and contain data arrays. In Figure 7 a visual representation of a tdms file graph is shown. Rectangles represent vertices in the file graph and diamonds depict datasets or properties [Nat21b].

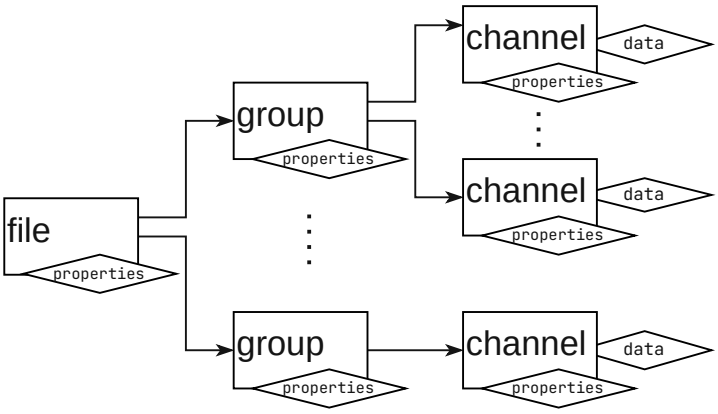


Figure 7: Visual representation of the tdms file structure

In the XBox-2 dataset groups contain data of one pulse of RF power. Each file contains all groups of a specific day of the experiment. Each channel contains data of a single sensor. On a single day of testing all pulses are recorded with the same sensors and therefore, in one file all groups contain channel data of the same origin. The channel properties list information of the corresponding sensors, group properties contain information of the time segments of their channels' data and file properties store the date of recording.

6.3.2 Event data

The event data contains the RF wave forms of single pulses. For each day a separate tdms file is created. Each event data file contains multiple groups. In

each group the segmented data of one pulse of the RF cavity is stored. In the XBox-2 test stand the pulse frequency is set to 50Hz which means that each 20ms a pulse of RF power is delivered to the cavity. The total recorded segment of time has a length of 2µs.

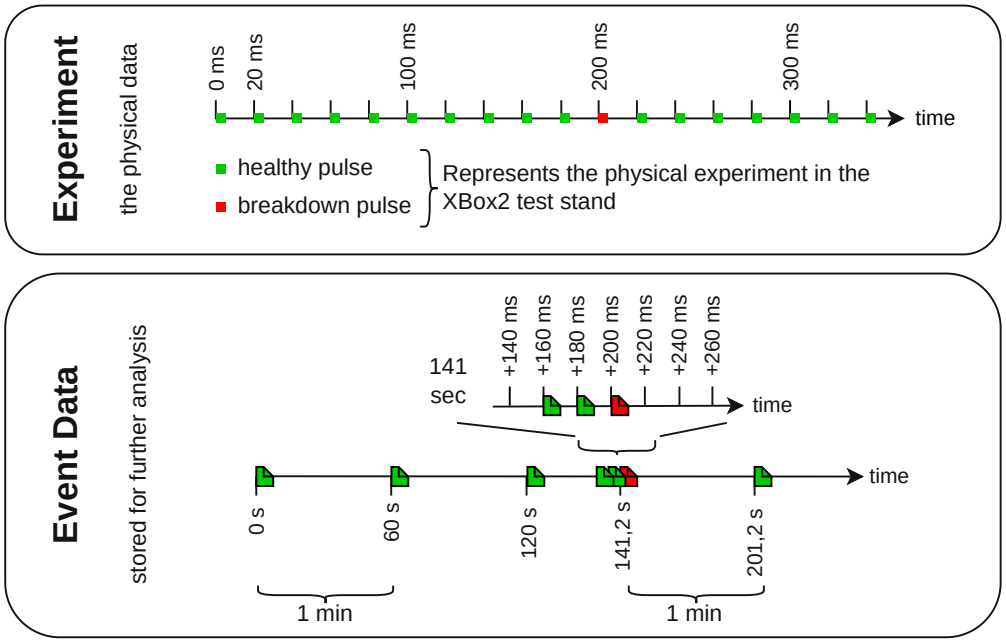


Figure 8: Storage intervals of the event data

In the test stand electric sensors are attached at different positions. The amplitude of the incoming RF wave and the reflected RF wave are measured right before and after the structure. In Section 6.2 a more detailed overview of the recorded signals is given. The signals are delivered to the acquisition cards in the NI chassis via coaxial cables. For the event data the acquisition cards NI-5772 and NI-5761 with a sampling frequency of 1.6GS/s and 250MS/s are used to digitize the sensor signals. These digitizers scan the signal at discrete points and convert those readings to double precision values. Those values are then further processed in *LabVIEW*, a software for labs developed by NI [Nat21c].

In each event data group there are 16 channels with 3200 or 500 data points. This translates to the maximal sampling frequency of the above mentioned acquisition cards in the given time window of 2µs. The $3200 \cdot 8 + 500 \cdot 8 = 29\,600$ values with 16 B each have a total size of 473.6 kB.

In Figure 6 some of the event data time series are displayed. The most reliable signals for breakdown detection are the measurements of the up and down stream Faraday cup in Figure 6 (e) and (f). These monitor dark current that results from

field emitted chard particles that are accelerated by RF waves (see Section 6.2). A breakdown event is labeled as such when a certain dark current threshold, assigned by experts, is reached [Luc+19][Kov10].

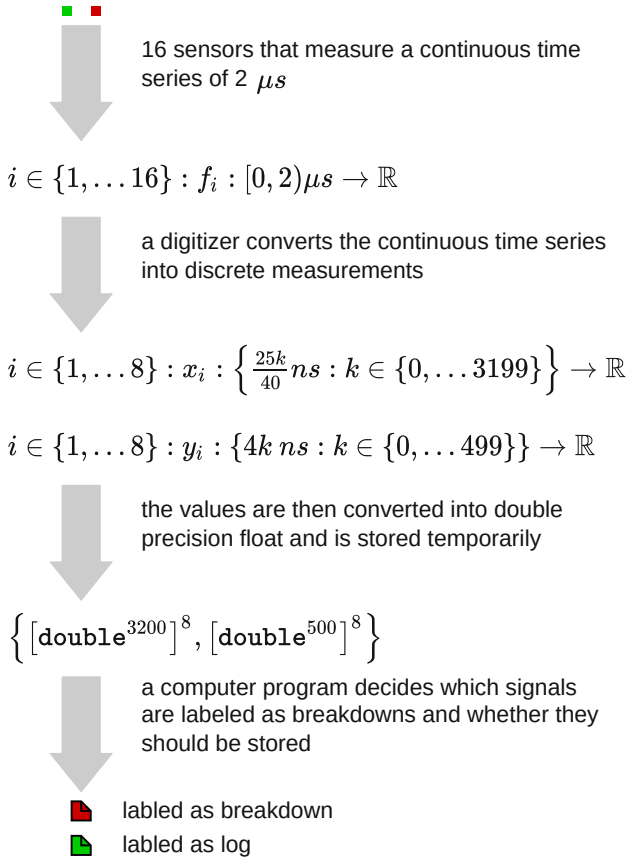


Figure 9: Data storing process from the physical experiment to the labeled data

Figure 8 illustrates the data structure of the event data. In the upper box every green square indicates a normal pulse of RF power. Each red square symbolizes a breakdown pulse(see Section 6.2). The data flow from the sensors in the accelerating structure to the tmds files is explained in Figure 9. Storing all $50 \cdot 3600 \cdot 24 = 4.32e6$ values would require $4.32e6 \cdot 473.6\,\text{kB} = 2.046\,\text{TB}$ which would overwhelm the system's capacities. Hence, only a fraction of the available pulses are stored.

The stored groups in the event data tdms files are separated in two different types, log groups and breakdown groups. When a breakdown in an RF cavity is detected the corresponding pulse is stored as a breakdown group. The two pulses prior to the breakdown pulse are stored as log groups for comparative purposes. Additionally,

a log group is stored once every minute in order to monitor regular operation. After each breakdown, the 60s saving cycle is restarted from zero.

6.3.3 Trend data

Trend data contains slowly changing data of the test environment. It describes a multi dimensional discrete time series that has one sample point every $\sim$ 1.5s. The exact storing intervals depend on the work load of the processor and thus vary. In general, the data stored in the trend data only measures slow changing values, also known as trends. In contrast, changes in the event data happen orders of magnitude faster. All values of one day, in total about 57 600 entries, are stored in a trend data tdms file spread over several groups. The used sensors and thus also the channels are the same for all groups and recorded days.

7 Machine Learning

Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets called input and output space. The hypothesis space H is a subset of the set of all functions from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$, i.e. $H \subseteq \{f : \mathcal{X} \to \mathcal{Y}\}$. The function $\mathcal{C} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ is called a performance measure or cost function. The main subject of machine learning is the minimization problem on $X \subseteq \mathcal{X}$:

$$\text{find } g \in H \text{ such that } \mathcal{C}(x, g(x)) \text{ is minimal for all } x \in X.$$

Every machine learning project starts with the data $X \subseteq \mathcal{X}$ and a performance measure $\mathcal{C}$. The data is separated into two disjoint sets called training set X_{train} and test set X_{test}. Searching for a good approximation of the minimizer of the performance measure $\mathcal{C}$ on X_{train} is called training of the machine learning model. It is considered successful, if the function does not only approximate the minimizing function of $\mathcal{C}$ on X_{train} but also approximates the minimizing function of $\mathcal{C}$ on X_{test}. This generalization of the minimization from the training set to the test set is considered artificial knowledge gain [Fis20, Section 3].

The hypothesis spaces used for machine learning contain parameterized functions. A parametrization uniquely identifies each function in H. Neural networks are defined in Section 7.2. They are parameterized by their weights and biases. A more detailed analysis of the parametrization of neural networks can be found in the Bachelor Thesis *FEM Triangulation for the Poisson Problem on Polygonal Domains with Neural Networks* by L. Fischl [Fis20]. Let the parametrization of the functions in H be given by R_θ where $\theta \in \mathbb{R}^\ell$ are the parameters and ℓ is the number of parameters. Then training is the process of altering parameters θ such that the function R_θ yields the desired output.

7.1 Training Techniques

Let us now discuss different types of cost functions $\mathcal{C}$. Let the function $h : X \to \mathcal{Y}$ be defined on the project data X. A machine learning project with a performance measure $\mathcal{C}$ that is given by $\mathcal{C}(x, g(x)) = d(h(x), g(x))$ is called a supervised model. The function d describes the distance between $h(x)$ and $g(x)$. For continuous output spaces $\mathcal{Y}$ a metric like the euclidean distance $d(h(x), g(x)) = \|h(x) - g(x)\|_2$ can be used, while a divergence like the cross-entropy measure is mainly applied for discrete output spaces. A successfully trained function $\hat{g} \in H$ is an approximation of the supervising function h in the hypothesis space which is the minimum of the cost function on the training set. A machine learning model with a cost function that does not have this structure is called an unsupervised model. These

models do not use an optimal output h for the definition of their cost function. In Section 8.4.2 a variety of unsupervised machine learning techniques are presented and applied on data of the XBox-2 dataset.

The most prominent application of supervised machine learning techniques is applied on input datasets $Z \subseteq \mathcal{X} \times \mathcal{Y}$, where the two parts are called data and labels. With the data $X := \{x : (x, y) \in Z\}$ and the cost function $\mathcal{C}$ defined with $\forall (x, y) \in Z : h(x) := y$ by $\mathcal{C}(x, g(x)) := \|h(x) - g(x)\| = \|y - g(x)\|$ a supervised machine learning project is built.

The machine learning model's output on X_{test} can be tested with the cost function. The performance on X_{test} is a measure of performance on the whole input space $\mathcal{X}$. Machine learning models that perform well on X_{test} are therefore able to extrapolate the information gained on the training set to new unlabeled data. A common example is the MNIST-database[1] of hand written digits where each image of a digit is labeled with the respective integer number that can be seen. The set X contains a vectorized encoding of the the fixed-sized images and the set Y their integer labels. A successfully trained machine learning model can extrapolate the labeling from the training to the test set. It real world use cases a machine learning model is then applied on new input data $\mathcal{X} \setminus X$ where new labels, integer numbers, are created. Note that these machine learning model based labels of new, yet unlabeled data can not be validated with a cost function. In Section 8.5 a supervised machine learning model is trained to detect faulty behavior (breakdown) of a the XBox-2 accelerator before it occurs.

7.2 Neural Networks

A common example of machine learning models are neural networks. The hypothesis space H consists of concatenations of affine linear functions and non linear activation functions. These activation functions $\sigma_k : \mathbb{R} \to \mathbb{R}$ are defined on $\mathbb{R}$ and extended to $x \in \mathbb{R}^n$ by element wise application $\sigma_k^{[n]}(x) := (\sigma_k(x_j))_{j=1}^n$. In this section two types of neural networks will be defined: dense feed forward networks and convolutional neural networks.

Dense feed forward networks consist of a series of affine linear functions $(l_i)_{i=1}^n$ with $l_i : \mathbb{R}^{L_i} \to \mathbb{R}^{L_{i+1}}$ and activation functions $(\sigma_i)_{i=2}^{n+1}$. This type of network applied on the input $x \in \mathbb{R}^{L_1}$ is calculated by $R_\theta(x) = \sigma_{n+1}^{[L_{n+1}]}(l_n(\dots \sigma_2^{[L_2]}(l_1(x)) \dots))$. Its architecture is given by the dimensions of its layer $(L_i)_{i=1}^{n+1} \in \mathbb{N}^{n+1}$. L_1 is the dimension of the input layer and L_{n+1} the dimension of the output layer. The networks are called feed forward because the linear function of each layer is fed only

[1] http://yann.lecun.com/exdb/mnist/

with the output of the previous layer. The parameters of the affine linear functions are called neural weights (matrices) and biases (vectors) and will be denoted by θ. During training these parameters are adjusted by a gradient descent based training algorithm. The gradient descent algorithm is a minimization algorithm that searches for the minimum in the direction of the steepest descent. For a given set of parameters θ the updated parameters in one training step are given by

$$\theta_{new} := \theta - \eta \nabla_\theta \mathcal{C}(x, R_\theta(x))$$

where $\eta \in \mathbb{R}^+$ is the step size and the negative gradient of the cost function is the step direction. Training ends when the gradient descent algorithm reaches a minimum. Improved versions of this algorithm, like the stochastic gradient descent, Nestrov momentum or the ADAM (Adaptive Momentum Estimation) algorithm, can improve training speed and the prevalence of getting stuck in local minima. Training parameters like the step size η have a huge impact on training outcome and are therefore called hyper-parameters. These improved algorithms increase the robustness of the training outcome to hyper parameters. A more detailed definition of neural networks and the calculation of the gradient can be found in [Fis20, Section 3]. State of the art training algorithms for machine learning are described in [GBC16, Section 8].

Convolutional neural networks (CNN) are used for time series data, image recognition or other multidimensional data where the order of the input data is an essential feature. Instead of a concatenation of linear functions with activation functions it contains three types of layers in the following order: convolution layers, pooling layers and fully connected layers. Images are specifically well suited for the application of CNN because the convolution layer processes blocks of neighbor pixels. This preserves local dependencies of pixels. The input space, the set of images $\mathcal{X} \subseteq \mathbb{R}^{n \times n}$, contains matrices that encode the brightness values of each pixel.

Dense feed forward networks have fully connected layers, i.e. in each layer every output value depends on all values of the previous layer. The convolution layer on the other hand uses local connectivity, where each output value only depends on few values of the previous layer. It is implemented with use of the discrete convolution of a kernel matrix $\varrho : \mathbb{R}^{m \times m}$ with $m \in \mathbb{N}$ uneven and $m < n$, which defines a kernel function $\varrho * \cdot : R^{n \times n} \to \mathbb{R}^{(n-m+1)\times(n-m+1)}$. The convolution of ϱ with a matrix $x \in \mathcal{X}$ is given by

$$[\varrho * x](i,j) = \sum_{\ell=1}^{m} \sum_{k=1}^{m} \varrho\big(\ell, k\big) x\big(i - \ell + (m+1)/2, j - k + (m+1)/2\big)$$

with $i, j \in \{1, \ldots, n - m + 1\}$. The pooling layers are designed to reducing the size of the matrix by a factor of $r \in \mathbb{N}$. The input matrix of this layer is separated

into blocks of size lower or equal to $r \times r$ and the maximal value, the mean value or the minimal value extracted. The pooling layer can be calculated by neural networks with a relu activation function. The fully connected layers consist of affine linear functions and activation functions similar to dense feed forward networks. The convolution, pooling, and fully connected layers are concatenated such the resulting neural networks describes a function from $\mathcal{X}$ to $\mathcal{Y}$.

8 Data Analysis Framework

8.1 Introduction

At CERN machine learning has proved to be a useful tool for data exploration and data analysis of experimental data and even created IML (Inter-Experimental LHC Machine Learning Working Group). It is, however, considered a novel technique in machine maintenance where its predictions might even improve machine reliability [Obe+22]. The main disadvantage is that preparing data to be processable by machine learning software is tedious and time consuming. The raw data is often uncleaned and stored in a unique data format that is dependent on the monitoring software. Individual solutions that are tailored to each set of raw data can not be used for other projects and lack the necessary transparency for collaboration. In every machine learning projects similar operational steps are applied to data preparation and machine learning. Consequently, a reusable, sustainable framework for machine learning can be adapted to multiple use cases and increase the implementation's transparency.

This section will start with the explanation of the steps a machine learning project goes through. Each step from raw data to the application of machine learning software will be discussed. In parallel, the framework will be tested on the example of the XBox-2 (see Section 6.2) dataset. The focus of the first part of this section is to test and decide on a data format that will be used throughout the whole framework for data transfer and manipulation.

8.1.1 Abstract Framework

The data analysis framework is a set of building blocks that serves as a flexible tool for data processing and machine learning. The main steps used in this framework are:

i) Transformation
ii) Exploration
iii) Modeling
iv) Execution

Every data analysis project may have a unique data format. Thus, as a first step, the data has to be transformed into a fixed data format so that the following building blocks that manipulate data universally applicable. In the second step

the data structure is explored, faulty data is filtered and its features (distinct properties like mean value, standard deviation, maximal value etc) are studied. Unsupervised machine learning can help to find outliers and to understand correlations. In the third step, supervised machine learning techniques are applied for predictions and its performance in generalization is tested. In the fourth step these analysis results are applied at the device under test. The implementation of a trained machine learning model to be used for predictions is an individual task. However, conducting a sensitivity analysis that investigates the decision making process is universally applicable for neural networks. It can potentially improve the understanding of the underlying data.

In Figure 10 the cycle of the machine learning framework is visualized. Each operational step of the framework is symbolized by a large arrow and each sub task by a small arrow. Data is passed on from one operational step to the next with newly created files that are symbolized by notes which are located in the corners of the figure. In Section 8.2 a selection of available data formats is analyzed. In Section 8.2.2 the most promising formats are compared.

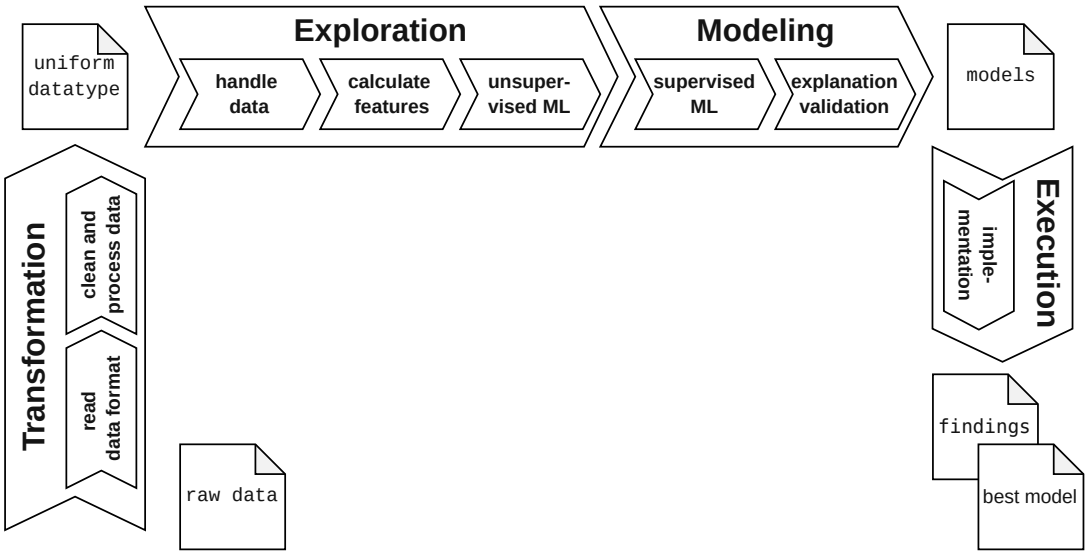


Figure 10: Machine learning framework flow diagram

8.2 Data Format

Research projects often use a unique data type, structure, and format that serve a very specific purpose. The data of every project has to be reformatted to the

same data type in order to reuse existing implementations for data manipulation and analysis. We call this process *transformation*. We choose the data type of the output to be:

i) easy to use in Python
ii) fast to read (single time series)
iii) processed quickly (sorted, filtered, calculated)
iv) transformable to arrays for machine learning

Machine learning research is developing rapidly. Therefore, it is beneficial to use programming environments that are on the fore front of research. Respective packages like Pytorch, Tensorflow and Keras all have good Python support. Consequently, we require the data format to be Python friendly. Analyzing time series sensory data with machine learning has high priority in the analysis of scientific data. It is important to read data from a single sensor for large sections of time quickly for analysis and feature calculation purposes. Computationally heavy calculations on these time series should be executed quickly. Machine learning algorithms generally require arrays, a list of values, as input data. Therefore, it is necessary that the chosen data can be transformed to arrays efficiently.

The packages Pytorch and Tensorflow each have their own data format torch and Tensorflow records that can store tensors of higher order. They, however are not directly interchangeable and therefore inhibit switching from one to another. Additionally, the context and additional information (like properties in tdms files) can not be directly stored in these tensor based formats.

8.2.1 Data Format Candidates

The four different data formats

i) Tdms Files
ii) Tabular Format
iii) Databases
iv) Hdf Files

are file formats that are analyzed in detail for this framework. In the following, an attempt to store the XBox-2 data in these formats will be done. This will lay the basis for an informed decision for the best suited data format for the data analysis framework.

Tdms Files The XBox-2 data set is originally stored in the Tdms File format. Instead of changing the data format, one could argue that most scientific data can be represented in tdms files. There is a package for Python called nptdms that can read and write tdms files. As mentioned above, tdms files consist of channels that are contained in groups that are contained in files. The data in channels can only be stored in one dimensional arrays. Converting higher order tensors into a tensor of first order, or array, requires either proprietary software by NI (eg. its image saving software) or custom conversion tools which would reduce the usability. Without such transformations the highest order object that can be represented by a tdms file is a tensor of third order, which limits its use cases.

The main disadvantage for data analysis is the internal structure of the tdms files. They are write oriented, which means that they are extremely fast at storing data. This is necessary in case of scientific studies at CERN because of the high resolution data and high frequency tests. Here, the tdms files need to be filled incrementally by the data streaming application in *LabVIEW*. During this data streaming process an index file is created. This index file is a copy of the original file, but instead of the raw data it only contains links to the data arrays. As long as this file exists navigation through the groups and channels can be done quickly as it only happens in the index file. When a dataset of a channel is selected, the link in the index file leads directly to the array in the main tdms file. However, only proprietary NI software can make use of existing index files. Other software, like the nptdms package, has to create its index files during every reading of the tdms file. This process of index file creation is called opening. Thus reading small segments of tdms files in Python can take orders of magnitude longer to read because the whole file has to be loaded in order to create this index file. Additionally, altering a tdms file with the nptdms package is not intended. One can overwrite the original file with the altered content as a workaround. However, this is resource heavy, time consuming, and tedious. In summary, it turns out that the tdms file format is optimized for fast write speeds in the lab environment but not suitable for post processing because of its limited data structure an write-optimized storage technique [Nat21a] [Ree20].

Tabular Format A tabular data format stores values on a two dimensional spread sheet, e.g. matrices. The main advantage of this format is its simplicity and the highly optimized software that is available. The proprietary `.xsl` or the xml based `.xslx` data format of the spread sheet program Microsoft Excel are also tabular data formats. In Python the package *pandas* has reached high popularity due to its ease of use combined with a powerful back end. Data can be loaded in pandas series or data frames that are either one or two dimensional data storage solutions [Pan21]. We will focus on pandas data frames as a representative of

tabular data formats.

The structure of the event data of the XBox-2 dataset cannot be directly transferred to a file format that can only resolve matrices (or higher order tensors). It contains not only the data arrays but also additional information that is stored in the tdms properties. For each pulse 3200×8 and 500×8 values have to be stored, which requires two separate tables. In addition, the channel properties and group properties require separate tables. Over a million files are needed in order to store all event data signals ($\sim 3 \cdot 10^5$) this way. This amount of files creates an unnecessary overhead by the file system. Consequently, other techniques are needed to store data of this complexity efficiently.

It is possible to fill cells with arbitrary objects such as vectors or matrices in some tabular data formats. Thus tensors of higher order can be stored. In case of pandas data frames this is not intended and unpractical as pandas automatically reformats given data to a table of scalars whenever possible. This feature is a compromise to ensure simplicity. For the event data of the XBox-2 data set, however, storing arrays in cells is possible as the array lengths are either 3200 or 500 which makes reformatting to a table of scalars impossible.

Pandas data frames cannot be stored directly, they have to be transformed either with a data serialization tool like pickle or by a transformation to another format like csv (comma separated values). The csv format converts float values to its decimal representation, converts a fixed amount of digits into strings and stores them in a text file separated by commas. Storing float values as strings of fixed size of its decimal representation has two main flaws. Firstly, the transition from float to a fixed digit decimal representation comes with a major loss of information. Secondly, characters with an 8 bit encoding can only use a fraction of its potential storing space to store numbers, the rest of possible storage space is wasted. Therefore, it is not recommended to use csv for scientific applications [Pyt21a]. Pickle is an external tool that converts any Python object like a data frame into a byte stream that can be stored directly in the memory. This byte stream can be converted back into a data frame. Note that pickle is regularly updated decreasing backwards compatibility with servers that run software which is not up to date. Nevertheless, storing pandas data frames with the help of the pickle protocol still is the best option available [Pyt21b].

In summary, tabular data formats can only store data with restricted complexity. For the XBox-2 dataset some restructuring of the raw data had to be done in order to fit pandas data frames.

Databases Databases are collections of data that are able to store highly complex data structures and their relationships. A database can be organized with a DBMS (data base management system). SQL is a DBMS that can create, manipulate and query databases. Other features like elaborate access control, security and automatic filling of the database are not relevant in this scientific use case. SQL is operated with a query language and its code differs significantly to that of classical programming languages.

Some databases do not provide data types that can store arrays. Separate tables with a column for each entry can be created for arrays of fixed size. However, the number of columns is constrained in many DBMS to a number smaller than 3200, which is the largest array length in the XBox-2 dataset [KA15, Chapter 4].

Databases are typically stored tuple by tuple, which is called row-oriented. Writing a tuple into memory is fast because the existing table only needs to be extended with one tuple. During the writing process the spot where the entry should end up needs to be determined and then the values of the tuples can be inserted. However, write speed is not the main concern in case of scientific data that is stored for analysis in retrospect. In fact, read speed is much more important. Reading a single attribute of a table that is stored row-oriented requires reading the whole table. Thus a read specialized DBMS like C-Store that stores data in columns is needed. When done correctly, selecting data on row-wise-stored data is much quicker [Sto+18].

Databases can handle almost any data structure, including the XBox-2 dataset. A detailed explanation how to store Tdms files in general as a database can be found in the appendix in Section 10.1. The main advantages of using a database and DBMS to store scientific data are the scalability and the possibility to perform highly complex queries efficiently. In particular, query languages are designed to manage large amounts of highly complex data. The main disadvantage, however, is that its creation, insertion of values, and writing of data selection queries requires the knowledge of query languages. There are Python packages, like sqlight3, that enable the control of a DBMS in Python code, but they are not easy to use as they still require the knowledge of query languages. Additionally, most available databases are row-oriented while only commercial databases offer the flexibility and speed that are needed for machine learning projects.

Hdf Files The hdf (hierarchical data format) is an open source file format that is based on a root tree structure. It is maintained by the hdf group, a non-profit organization and used in several scientific applications. There are different versions of hdf files, here the more advanced hdf5 files are used. The core processing is implemented in the programming language C and highly optimized, which con-

tributes to its processing speed. As discussed above, the framework is has to be implemented in Python because of the machine learning software that is available there. The package h5py for Python allows managing hdf files with a high level API (Application Programming Interface) and is distributed by the hdf group [Col+21]. The main objects in hdf5 files are groups and datasets. A special group called file is at the root of the tree based format. Datasets can only be leaves of the tree while groups can be inner vertices or leaves if they are empty. Datasets and groups can all contain additional information in so called attributes.

The hdf objects and their naming style works similar to the unix file system, which is familiar when programming on a Linux based system. Navigating through an hdf file can be done like navigation in directories. Access is possible directly by the full hdf path with a backslash in the beginning and separating backslashes in between group names or by calling groups incrementally. Similar to unix, hard and soft (symbolic) links can be attached at the tree. An internal link points to a group or dataset of the graph within an hdf file. Placing this link as a child of a group will lead the user to the destination group or dataset. The link acts like a shallow copy of its origin. A hard link copies the data location while a soft link copies its symbolic location inside the file graph. Similar to mounting external storage in the /mnt or /media directory in unix it is possible to link groups or datasets from external hdf files [Col+21].

The tdms file format is a tree based file format with additional properties at each node that is bound to depth two. Thus a tdms file can be directly converted to the more general hdf5 format that can store any tree based structure. During the translation process each tdms file object is converted to an hdf root group, each tdms group to an hdf group and each tdms channel to an hdf dataset. In each step all tdms properties can be stored as hdf attributes. This conversion is already implemented in the nptdms package that is used for reading tdms files in Python.

8.2.2 Data Format Comparison

In the following an empirical comparison of the data formats nptdms, hdf5, and pandas data frames is done empirically by the example of the XBox-2 data. The decision of the data type used for the data analysis framework is based on this comparison. The objective of the test is to see which data format allows reading all available data from a few sensors fastest. Similar reading of data is essential for data analysis. During the comparison the read speed of 1, 3, 15 and all 35 of 35 available channels is tested. Additionally, the amount of storage space that the data needs is monitored. During the test all trend data is converted to hdf5 files with the conversion tool available in the nptdms package. The trend data is

converted to pandas data frames so that each channel data is located in a separate column. The data frames are stored with the pickle protocol for data serialization. The results of the speed test are presented in table 1.

	nptdms	pd.df+pickle	hdf5
space (GB)	20.5 GB	2.8 GB	2.8 GB
read (TD 1 channel)	$\sim$ 60 min	4 s	0.5 s
read (TD 3 channels)	$\sim$ 60 min	4 s	0.7 s
read (TD 15 channels)	$\sim$ 60 min	4 s	2.3 s
read (TD 35 channels)	$\sim$ 60 min	4 s	6.0 s

Table 1: Read speed of tdms, pandas data frames and hdf5 files in Python

Data streaming in tdms files leads to very inefficient use of storage space and very slow read times. This effect is more noticeable with trend data. This difference might be caused by the writing process. In trend data one value is added every $\sim$ 1.5s in each channel, which leads to approximately 57 600 write processes spread out during the day. In contrast to that fewer large chunks of data are written at once in event data. The storage space requirement is reduced by 86% by converting the trend data from tdms to a different file format.

Read time of pandas data frames does not change with the number of channels to read because pickle cannot de-serialize single columns. In comparison, hdf5 can make use of the reduced amount of data to read because of its tree based file structure. Reading a few channels with hdf5 takes under a second while reading all channels takes 6 seconds. Reading tdms files without the index file, which contains the file structure, requires an opening process. The file structure is analyzed and the index file is created by reading all data into memory. This method seems to be highly inefficient as the reading time is thousands of times higher than that of hdf5 files or pandas data frames.

In order to save additional space, data compression tools can be used. One of the most common compression algorithms is gzip (gnu zip), which is based on Huffman coding combined with additional compression of repetitive data. Encoding and decoding is computationally expensive which is a trade-off for the decreased storage space. File compression can be applied on different layers of a file format. Applying gzip on a whole folder or file is considered a top layer application. Alternatively, the compression of hdf5 files can be applied on each individual dataset automatically, which is considered a bottom layer application. A comparison of different compression strategies can be seen in table 2.

In all cases an increase of read time can be observed with compressed data. Top level compression leads to a constant decompression time added to the whole

	pd.df+pickle+gzip		hdf5		
using gzip	n	y	y top	y top	y bottom
space (GB)	2.8 GB	1 GB	2.8 GB	1 GB	2.9 GB
read (TD 1 channel)	4 s	12 s	0.5 s	12 s	0.5 s
read (TD 3 channels)	4 s	12 s	0.7 s	13 s	1.5 s
read (TD 15 channels)	4 s	12 s	2.3 s	14 s	5.5 s
read (TD 35 channels)	4 s	12 s	6.0 s	18 s	15.0 s

Table 2: Comparison of compression strategies on pandas data frames and hdf5 files

read time. This is a consequence of the time needed for the decompression of the whole file before a section can be read. A noticeable decrease in read time for fewer channels can be achieved by the hdf5 internal compression algorithm. Compressing each dataset, however, does not decrease the storage space. No repetitive parts can be compressed when applying the gzip algorithm individually for each dataset. This shows that data compression can decrease the total storage space. More repetitive data compressed at once on a higher layer leads to a better data compression rate while reading time for small segments of data benefits from the compression on a lower layer.

In summary, the hdf5 format has faster read times for smaller segments of data compared to pandas data frames, which is an advantage for machine learning. Additionally, hdf5 offers more flexibility in compression algorithms compared to tabular data formats. The tdms format is inadequate for data analysis and databases require the knowledge of query languages, which is an obstacle when working in Python with this framework.

8.3 Transformation

The main task of the transformation step is the conversion from the input data format to the final data format used in the following steps of the framework. During that step data that does not fit a pre-selected structure is filtered. As analyzed in Section 8.2.2 on the example of the XBox-2 dataset, the format most suited for the conversion is hdf5. The framework should also be flexible enough to be used for another data format. Consequently, the conversion is organized with a class structure that supports the conversion of various input and output formats.

In Figure 11 a visual representation of the transformation is depicted. In the top left corner the main program is shown. It is implemented to convert XBox-2 data from tdms to hdf5 format. The gray arrows point to the `Convert` class, which is

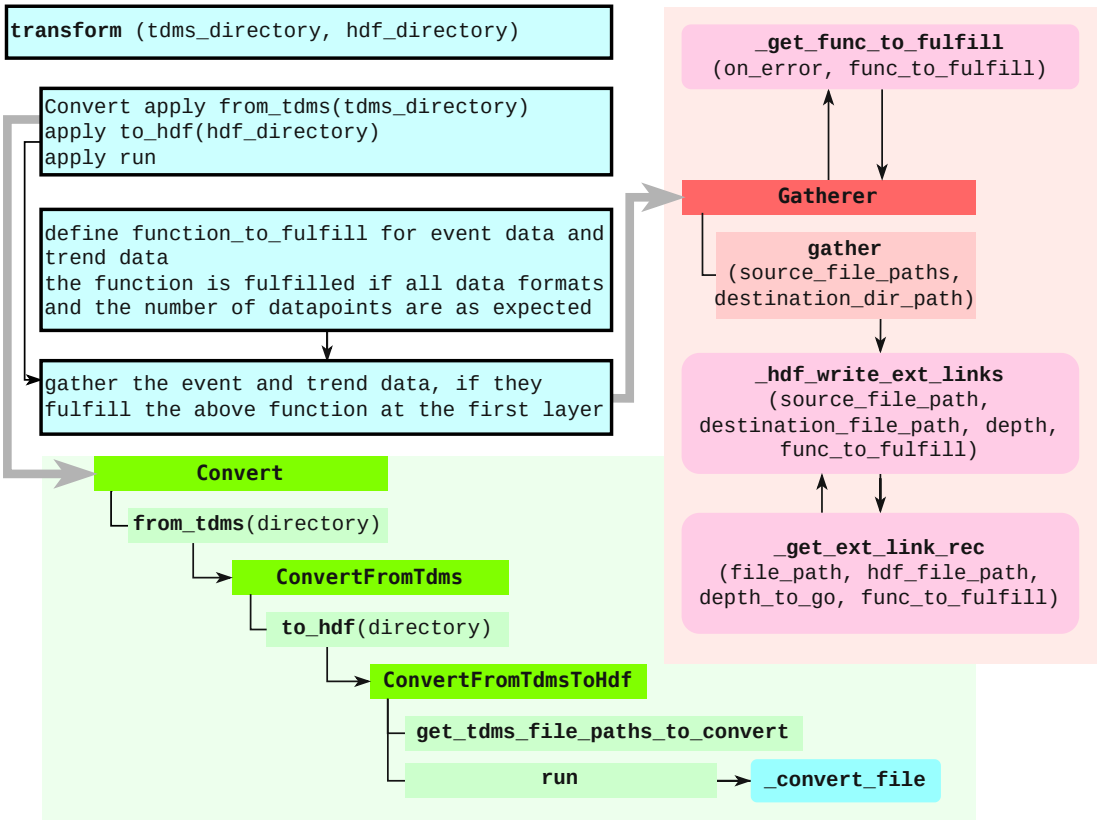


Figure 11: A visual representation of the implementation of the transformation

located on the bottom in green and the `Gatherer` class which is on the right hand side in red. Both of these classes can be extended to be applied on different data formats.

8.3.1 Convert Data

The class structure for conversion consists of three layers.

The `Convert` class is the base layer with the attributes `check_already_converted` and `num_processes`. The first one defines whether the program should look for already converted files or whether all files of the input directory should be converted. The second input parameter defines how many processing cores should be used in parallel for the conversion.

The preceding class layer defines the input directory, in the case of the XBox-2 dataset a directory of tdms files. The class name is `ConvertFromTdms`. An instance of this class layer can be created with `from_tdms`, a function of the `Convert` class.

When an additional data format has to be transformed by the `Convert` class, an additional preceding class is added.

The final class layer defines the output data format. As suggested in Section 8.2.2, hdf files will be used as they are the most suited data format for the XBox-2 dataset. An instance of the `ConvertFromTdmsToHdf` class can be created by the `to_hdf` function from the previous class layer. The conversion process of all files listed for conversion is started with a call of the function `run`. It calls the hidden function `_convert_file` which opens each tdms file and converts it. An additional output data format can be added to the `ConvertFromTdms` class by an additional subsequent class.

Thus an additional input data format requires the implementation of another second and third layer class and hidden file converter. Another output data format would only require a new third layer class and hidden file converter.

The class structure is chosen so that its class structure's application is convenient and self explanatory during the call in the main transformation program. Class attributes in Python are called with a dot between attribute and class member. Let the input directory be given by `tdms_dir` and the output directory by `hdf_dir` then the call of the conversion with 4 parallel processes with a check of already converted files can be done with

```
Convert(check_already_converted=True, num_processes=4).
from_tdms(tdms_dir).
to_hdf(hdf_dir).run()
```
[line 26-28 Listing 3]

8.3.2 Clean and Process Data

In the lab environment of the XBox-2 test stand adjustments of the machine had to be made during testing. For example, from time to time electronic components had to be moved and cables unplugged or rearranged. The recorded data during periods of such changes miss certain data points. Consequently, fixed data types and structures are defined that standardize all event and trend data. A detailed description of the filter criteria is defined in the course of this section and its implementation can be found in the Appendix in Listing 3. The filtering is done by functions that return the Boolean value true if it fulfills the standardized structure and false if not. Additionally, the files are restructured to improve usability.

The hdf format enables restructuring data of multiple hdf files into one with external links. Instead of copying all data into one large file, known as deep copy, only the file path of the hdf file and its internal hdf path are stored, which is known as shallow copy. The `Gatherer` class can gather groups of a given hdf layer of a list of

hdf files into one hdf file if these groups fulfill the `function_to_fulfill`. The layer of a group is given by its depth in the file tree graph starting from the root. The output hdf file is given by the destination file path. Additionally, with the option `on_error` one can define what should be done when the `function_to_fulfill` fails execution and thus throws an error. The gathering with external links keeps the size of individual files small, which is necessary for file transfer in network environments where the maximal shareable file size is limited.

The `ed_function_to_fulfill` tests each recorded pulse stored in a group of the event data if there are 8 channels with a length 3200 and 500 respectively. The tdms format supports storing NaN (Not a Number) and inf (infinity) values. These values are of no use in further analysis and are labeled smelly. Smelly values of event data are filtered. Its implementation can be found in lines 41-68 in Listing 3.

The `td_function_to_fulfill` tests each group of the trend data tdms file. It checks if there are 35 channels and if all channels have same lengths. An implementation can be found in lines 29-35 of Listing 3.

For the XBox-2 dataset trend data and event data are gathered separately with the help of external links. In the event data each group contains the data of one pulse of RF power that is sent to the RF cavity. In the `Gatherer`, data is joined at the first layer so that values of all pulses of all days are combined in one hdf file. The trend data is also gathered at its first layer for all days which makes the combined file a list of truncated sections of continuous data. The application of the `Gatherer` for the XBox-2 data can be seen in lines 37-39 and 70-72 of Listing 3.

The trend data is a consecutive time series with equidistant intervals between measurements. It is stored, however, as segments of time series. Combining all trend data requires merging datasets that are segmented in several groups placed in several trend data files. The hdf file that contains all merged datasets is a deep copy of its segments. Filtering faulty values is different in trend data. In event data a fixed amount of event data values is expected for data analysis. In trend data, however, each data point is recorded and analyzed individually. Therefore, if a single data point is faulty it can be removed or altered individually. This filtering procedure is done *by row*. An implementation of `merge` and `clean_by_row` can be found in lines 20-36 and 81-115 of list 2. Furthermore, the merged data can be sorted by a specific column. In case of the XBox-2 data, it is sorted by its timestamp. The sorting function is implemented in lines 118-129 of list 2. These data manipulation functions are called in the main program (see list 1) right after the transformation.

8.4 Exploration

The main task of the exploration step of the data analysis framework is feature calculation, also known as data handling, and an exploration of the data with unsupervised machine learning. Firstly the context data file is created. It contains additional values called features that describe characteristic properties of the experiment. It can be either data of its surrounding environment or statistical properties calculated from data arrays. The most common examples of statistical features are length, mean, median, maximal and minimal value, variance or Fourier coefficients when dealing with time series. Secondly, the exploration with unsupervised machine learning helps to understand inner relations of the data and to find statistical outliers.

8.4.1 Feature Calculation

The main point of interest of this analysis is the occurrence of breakdowns, which is information that is stored in the event data. The hdf file `context.hdf` is created in an effort to keep the original data measurements unchanged. The context file contains features for each event. It contains event data attributes, features calculated form event data and the previous trend data point.

The storing process of event and trend data happens completely independently. The information of breakdown occurrences is only contained in the event data. Therefore, using the trend data for analyzing breakdowns is only possible combining the two datasets. Timestamps in trend and event data enable joining the recordings. It is important, however, to keep the correct order of events, otherwise a prediction might be deceptive when it uses data that is subsequent of a breakdown. A timestamp is stored in the group attributes for each pulse of the event data while in trend data timestamps are stored in a channel. Attributes are read during the feature selection process. Therefore, merging of event and trend data timestamps has to be done after they are read. An implementation of the combination of event and trend data can be found in the function `manage_trend_data_features` in lines 72-89 of list 6.

The statistical properties of event data are part of the context data file. The feature calculation and the process of storing these values to the context data file is implemented separately. The essence of its implementation is depicted in Figure 12. A legend of the used shapes is located in the lower left corner. The program `XBox2ContextDataCreator` manages the feature calculation and writing of the context data file. It is called in the main program in line 49-51 of Listing 1. In the left upper corner the `feature_class.py` is located in red. It contains the class structure of all custom features. Each trend-data, event-

data, and event-attribute feature is created individually by their `get_` functions that are located in the center in purple. Furthermore, the `get_tsfresh` function automates the use of the feature library. The created feature calculators are used by the `XBox2ContextDataCreator` and kept in temporary storage. The `Context_Data_Writer` has two sub classes. The column-wise-writer writes one feature of all events at once while the row-wise-writer writes a bunch of features of a single event at once.

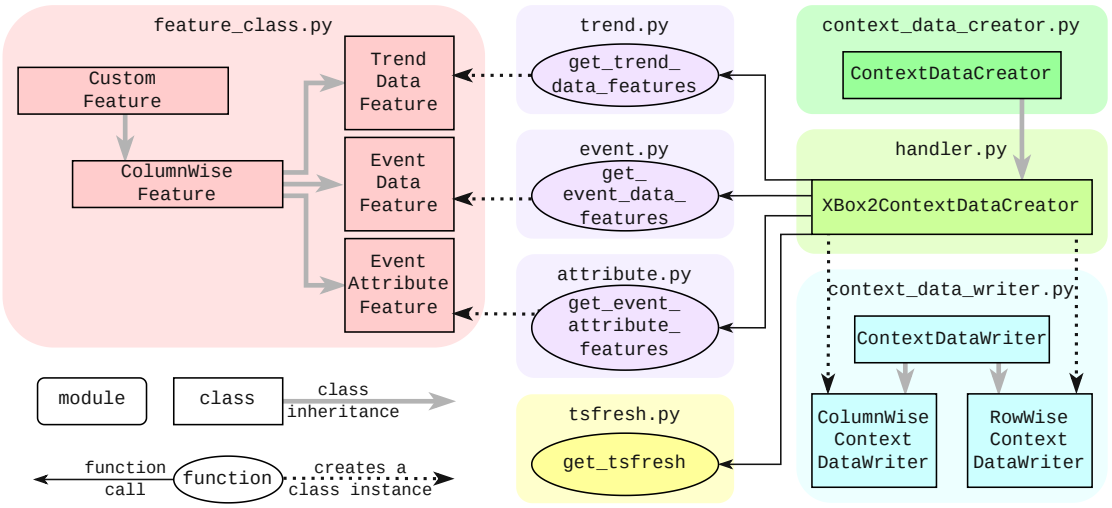


Figure 12: Visual representation of the class structure of the feature calculation and context data writing process

Event Attribute Features Firstly, the event attribute features are created. During this process all group attributes (originally tdms properties) of each event are stored in a temporary vector. At the end of the calculation of this feature the temporary vector of attributes is written into the context data by the column-wise-writer.

Trend Data Features Secondly, trend data features are created. They require the exact timestamps of the event data, which are already stored in the event data attributes. Finding the right trend data points is fast because of the combined trend data that was prepared during the transformation step.

Event Data Features Finally, the event data features are computed. The calculation of general statistical properties is done with the help of a package called *tsfresh*, which offers implementations of feature calculations [BC]. The number of

features that are calculated can be altered between basic (9 values), efficient (∼800 values) and comprehensive (∼2000 values). Additionally to traditional statistical features the efficient and comprehensive sets contain feature functions that depend on additional parameters and therefore yield multiple values at once. Feature functions like *k-quantil*, *count values above x* or *Fourier-coefficients of frequency ℓ* allow for an increased number of features. All other features are implemented as a `CustomFeature` specific to the XBox-2 dataset.

The reading process takes up most time during event data feature calculation. Therefore, each event data feature is only read once and immediately calculated by both types of features, tsfresh and custom features. The custom event data features are written into temporary vectors and finally written by the column wise context data writer. Consider one thousand tsfresh features that each take up 8B of storage space. Then the required temporary storage space for the 297 128 events with data of 16 sensors each would be $297\,128 \cdot 16 \cdot 8\text{byte} \cdot 1000 = 38.03\text{GB}$, which would create unnecessary overhead. Consequently, the row-wise-writer is used for tsfresh features. It immediately writes all features of a single event into the context data file right after its calculation.

The feature calculators and their destination locations in the context data file are depicted in Figure 13. On the left hand side the step by step calculation of features is illustrated. Each arrow points from a feature calculator, placed on the left hand side, to its destination datasets of the context data file, which can be seen on the right hand side. Event attribute features are stored as direct children of the context data file, whereas the datasets of tsfresh and custom features are stored as children of their separate sensors' data groups, which in turn are children of the file. The trend data values of the closest preceding records are located in a separate group called `PrevTrendData`.

8.4.2 Unsupervised Machine Learning

Dimension reduction is a process that defines a function that projects a high dimensional dataset to a lower dimensional space while preserving the essence of the dataset. The essence of a dataset is defined as the similarities that they have or the differences that distinguish them. The lower dimensional representation can be used to get an insight into the essential properties with a visual representation of the projection. The projection can be applied on new data that was not used during its creation. Generalization is achieved if the resulting low dimensional representation also displays connecting and differing properties. Therefore, dimension reduction can be used as a machine learning tool with unsupervised training (see Section 7).

The simplest projection function for unsupervised machine learning is a linear function. It, however, can only preserve linear properties of its training data.

Neural networks with a special architecture can be used to define a projection function. Consider a neural network with multiple layers that contains same in and output dimensions and a hidden layer with only a few neurons called restriction layer. This network is trained to represent the identity function on a given dataset. If possible it is able to *encode* the input information in the restriction layer and *decode* it again afterwards. This type of neural network is called *autoencoder*.

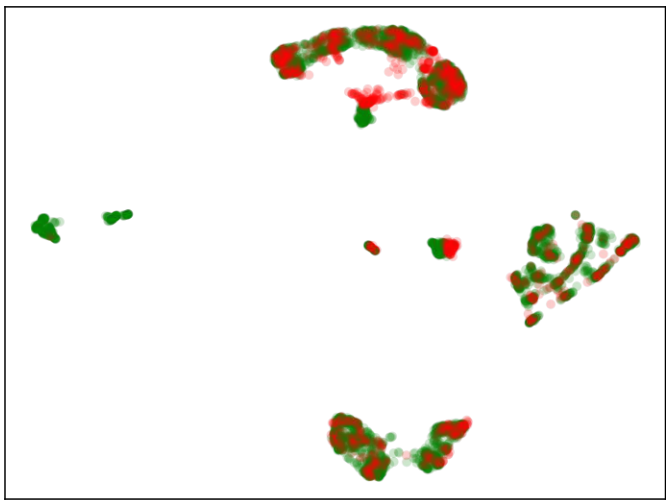


Figure 14: UMAP projection of pre-breakdown events in red and healthy events in green

Statistical graph based methods that search for a manifold in the high dimensional input space with a lower dimensional representation have been highly successful in machine learning. The most common examples are TSNE (t-distributed stochastic neighbor embedding) and its improved version UMAP (uniform manifold approximation and projection) have gained high popularity. The UMAP algorithm works in three major steps. Firstly, it computes a weighted graph that resolves the high dimensional relations of the data. A relational graph is defined with help of a directed graph where each point is connected to its k-nearest neighbors. Each directed edge is assigned with a weight equal to its relative distance in relation to the rest of its k-nearest neighbors. The directed weighted graph is turned into a weighted graph by combining both directional weights. Secondly, UMAP initializes a low dimensional representation and calculates its relational graph. Thirdly, it improves the lower dimensional representation such that it's relational graph most accurately represents that of its higher dimensional pair. This optimization

process uses stochastic gradient descent to minimize an improved cross entropy measure of its graph edges [MHM18].

An analysis of XBox-2 data with TSNE is done in Chapter *III. B. Exploration* by C. Obermair [Obe+22]. We use the UMAP algorithm for unsupervised analysis of the XBox-2 dataset. The previous trend data signal and the max, min, mean, median, standard deviation, sum and variance of each event data signal are used for the analysis. All data is normalized to have mean zero and standard deviation one before the training algorithms are applied.

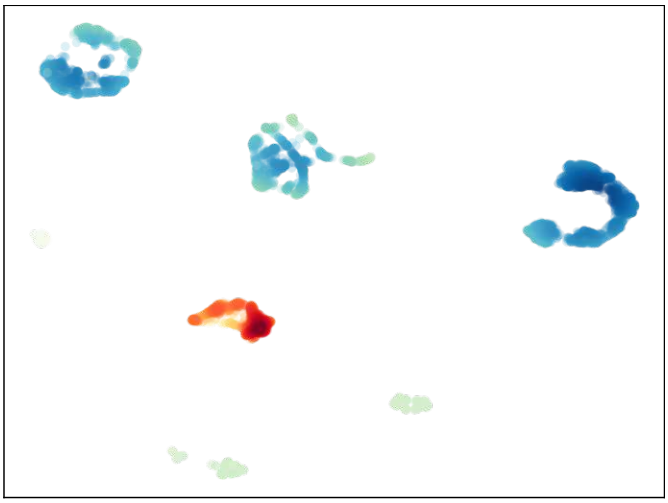


Figure 15: UMAP projection of breakdown events in orange-red and healthy events in green-blue, stacked points are indicated by a darker color

Firstly, only pre-breakdown events where a breakdown happened 20ms afterwards and some of the healthy events were used for analysis. In Figure 14 the UMAP projection is displayed. Each projection of a pre-breakdown event is marked in red and each projection of a healthy event in green. We can see that no clear distinction between pre-breakdown and healthy pulses is possible by unsupervised training with UMAP. Therefore, no generalization can be done. The data points of the clusters do not indicate a clear distinction with respect to time.

Secondly, breakdown events and part of the healthy signals are analyzed. The UMAP projection onto the 2 dimensional plane is visualized in Figure 15. The breakdown signals are colored in shades of red and the healthy signals in shades of blue. The UMAP algorithm can distinguish breakdown and healthy clearly even though no labels are used during the definition of the projection. The darker the color the more points are stacked on top of each other.

This shows that breakdown and healthy data already is clearly separable in the

high dimensional space. A breakdown data event, however, contains the data of a vacuum arc that has already formed, thus no predictions are possible with this distinction.

Thirdly, the UMAP algorithm is applied on a random set of only healthy events. The results in Figure 16 are colored by runs. The segments of stable operation are numbered as run 1 to run 9. Runs are time segments where no settings are changed. This projection is able to separate events with respect to runs with very few outliers. However assigning events to distinct time intervals does not help for the detection of breakdowns.

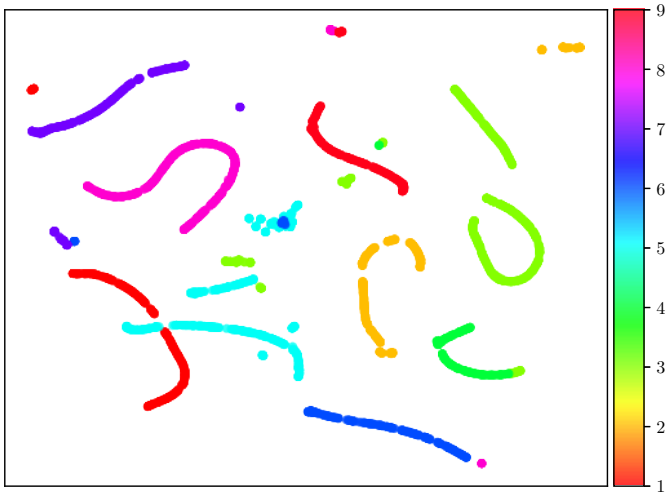


Figure 16: UMAP projection of healthy XBox-2 data colored by run

8.5 Modeling

During the modeling phase the data for analysis is selected and a machine learning algorithm is trained for breakdown prediction. The model's generalization is then tested on a test set or could even be integrated in the lab environment. Additionally an analysis of the machine learning model's predictions gives insights to the decision making process that can potentially improve the understanding of the underlying data.

8.5.1 Supervised Machine Learning

For each event of the event data 8 statistical properties of its data arrays and the first preceding trend data record are used for analysis, which is a total of 191 values per event. The number of healthy signals is disproportionately large

compared to the breakdown events. Only a random part of healthy events is chosen for training. The pre-breakdown events, where a breakdown happens 20ms afterwards, are selected for training. A distinction between healthy signals and pre-breakdown ones will enable a model to predict that a breakdown happens in the subsequent event. The labels used for supervised training are: *is a healthy event* and *is a pre breakdown event*.

A fully connected feed forward neural network with 2 hidden layers is chosen as a model. The two hidden layers have 191 nodes each, while the output layer only has a single node. The activation function in between the hidden layers is the relu (rectified linear unit) which is given by $\texttt{relu}(x) = \min(0, x)$. The sigmoid function, $\texttt{sig}(x) = (1 + \exp(-x))$ is used after the output layer. Its range is in $(0, 1)$ so the output is interpreted as a probability of its Boolean value 0 or 1.

Before training each dimension of the available data is scaled to have mean zero and standard deviation 1. The data is split into training and test data with a ratio of $\frac{1}{5}$ to $\frac{4}{5}$ randomly. During training validation data is separated from the training data to measure the performance of the network after each round of training. The Adam (adaptive moment estimation) algorithm with batches of size 10 is used for training. One round of training where all data is used is called epoch. The results were evaluated after 20 training epochs in this machine learning example.

During the evaluation of the model all output lower than $\frac{1}{2}$ is interpreted as a prediction of a healthy event and all values greater or equal to $\frac{1}{2}$ as a prediction of a pre-breakdown event. The outcome of the prediction is a binary event. We will consider the classification of a pre-breakdown event a positive and a not pre-breakdown a negative event. The goal of the machine learning model is to accurately classify actual positives and actual negatives as such.

The XBox-2 data is highly unbalanced which requires a special performance measure for predictions. Normally the accuracy, which divides the number of true predictions divided by the number of total predictions, is used. For imbalanced data, however, the accuracy is not suited to test the performance. Instead, the balanced accuracy is used. It is calculated by the mean of sensitivity and specificity.

$$\text{balanced accuracy} = \frac{\text{sensitivity} + \text{specificity}}{2} = \frac{1}{2}\left(\frac{\text{true pos.}}{\text{actual pos.}} + \frac{\text{true neg.}}{\text{actual neg.}}\right)$$

Sensitivity is the number of true positives divided by the number of actual positives and specificity is the number of true negatives divided by the number of actual negatives. The balanced accuracy is suited to evaluate the performance of a machine learning model on imbalanced data.

A k-fold cross validation splits the training data into k equally large parts. It does

k different training runs with $k-1$ parts used for training and one part for testing. The average performance is then measured separately with the balanced accuracy.

The machine learning model is trained on 6352 events of which 953 are labeled as pre-breakdown events. On the 5-fold cross validation test set there are on average 157 pre-breakdown events of which 150 are predicted correctly and 1113 healthy events of which 1072 are predicted correctly. The machine learning model is able to distinguish healthy events from pre-breakdown ones with a balanced accuracy of $95,88\%$ with a standard deviation of 1.21%. This was tested with a 5-fold cross validation. An implementation of the machine learning model can be found in line 55-99 of Listing 1.

This model suggests that breakdown occurrence is predictable. If settings in the accelerator can be changed shortly before a breakdown occurs it can potentially be prevented. Further verification is necessary before the machine learning models can be applied in the lab. A detailed discussion of results and comparison of different neural networks is done in Chapter III and IV by C. Obermair [Obe+22].

8.5.2 Explanation and Validation

The `shap` package is a sensitivity analysis tool that enables the exploration of the impact of each input feature named after Lloyd Shapely. At first, it calculates shapely values for each input data point. These values are a balanced measure of the impact of each input dimension on the output magnitude of the neural network. The mean of the shapely values then quantify the average impact of an input dimension on the predicted output [LL17].

sensor name	shapely value
internal pressure upstream	0.098
internal pressure before structure	0.0141
internal pressure load	0.00773

Table 3: The three most important trend data signals for breakdown prediction and their shapely values

Table 3 shows the three highest mean shapely values of the neural network trained in Section 8.5.1. All three signals describe the internal pressure at: upstream (placed at the beam axis of the structure, stream is the direction of the RF wave), before structure (in the wave guide leading to the structure), and load (RF terminator after the structure). For a more detailed description see Section 6.2 or [Obe+22, Chapter II]. The shapely values suggest that upstream internal pressure sensor is by far most responsible for the breakdown prediction. Its mean shapely

value is seven times higher than that of the second most important signal. As mentioned in Section 6.2 accelerator scientists observed that small deformations in the material of the accelerating unit create field enhancements in that region. It is believed that spikes in electric field strength melt material in the region of the deformation and thus disturb the vacuum[Nav17, Section 2.3]. The analysis of the machine learning model suggests that a disturbance of the vacuum is measurable much earlier than the breakdown occurrence.

8.6 Execution

The application of the knowledge obtained by the machine learning models is called execution. The transfer of results can be done either by using the trained machine learning model in the lab environment or by deriving information from the trained models.

The XBox-2 test stand is controlled by the lab software *LabVIEW*. The implementation of a pre-trained machine learning model is possible. Clean data is required for flawless operation. However, this is not always available. The implementation of a model that continues learning during operation is resource heavy. Furthermore, not understanding the reason for its predictions does not improve the understanding of the processes that cause breakdowns. Consequently, alternative ways of knowledge transfer are are required. This can be achieved by investigating the decision making process of the neural network as done in Section 8.5.2.

The XBox-2 test stand was already deactivated at the time these results were available. Therefore, it was impossible to test the results on the original machine. The succeeding model, the XBox-3 test stand, is similar and was in place at that time. The hypothesis is that breakdowns occurs after a vacuum disturbance which is monitored by the pressure sensor that is responsible for the prediction in the machine learning model. During the experiment the hypothesis is tested by changing the settings when a fluctuation in the pressure signal is detected. When the settings are changed back, the occurrence of breakdowns suggests that local deformations of the material cause the breakdown. The time available for experimental proof of the hypothesis on the XBox-3 accelerator was only 4 hours. 15 breakdowns occurred in 5 groups during that time. Two groups of breakdowns support the hypothesis and three do not. No conclusive results were possible due to the limited time for testing. Further experiments with a sufficient amount of breakdown data are needed to be able to verify this hypothesis. Additionally, an improved strategy for hypothesis testing is recommended.

In summary, the analysis of the machine learning model with shap from Section 8.5.2 suggests that a pressure signal disturbance is an indicator for break-

downs. No conclusive experimental proof was conducted during the testing of the failure hypothesis. However, the application of machine learning analysis gave new insights in breakdown occurrence in the XBox-2 accelerator. A more in-depth analysis of the experimental is done in Chapter IV.C. *Experimental validation* by C. Obermair [Obe+22].

9 Conclusion

In this thesis the standard model of particle physics is introduced. It is a model for the fundamental building blocks of matter. On the one hand it defines the objects investigated in particle physics, on the other hand it explains the need for high energy experiments. The axiomatic introduction of relativistic mechanics and electromagnetism allows a deeper understanding of the physics in particle accelerators. A theoretical comparison of collider and accelerator designs and collision strategies highlights the challenges of future accelerators. Calculations in the first part of the thesis show that colliding beam experiments are preferred over fixed target collisions. The main constraint of circular colliders are the magnets that bend the particle beam. In this thesis, the minimal required magnetic field strength needed to achieve collisions of 13TeV in the LHC (Large Hadron Collider) is calculated. This indicates that higher energy beam collisions are not possible with the currently available superconducting magnets. Therefore, a bigger circular accelerator is needed for higher energy experiments. New research goals can also be pursued with a linear accelerator which achieves higher luminosity, an important metric of particle colliders. The CLIC (Compact Linear Collider) is a linear collider designed by CERN and intended to be built in three stages within the next three to four decades. An experimental RF (radio frequency) cavity structure, called XBox-2, is introduced. It investigates the main boundary of this type of accelerators: breakdowns. In the second part of this thesis breakdown data of the XBox-2 test stand is analyzed.

Data preparation for machine learning is a resource heavy, time consuming and tedious process. In this thesis a robust and reusable framework is built that seeks to achieve high respectability and efficiency. The implementation of the framework is described with the support of graphical illustrations. In parallel, it is applied on data of the XBox-2 accelerator for the investigation of breakdowns. Firstly, the optimal data type used for data analysis is determined by an empirical study of a selection of available options: tdms files (used by LabVIEW by NI), tabular data formats (e.g. pandas data frames, excel, CSV), databases (e.g. SQL), and Hdf files (Hierachical data format). A transformation class structure, that is adaptable for different data types, is implemented. Secondly, data is explored by feature calculation and analysis with unsupervised machine learning techniques. An unsupervised algorithm was able to distinguish events with respect to different operating parameters. However, no breakdown predictions were possible with unsupervised training. Thirdly, neural networks are used for supervised machine learning. A trained model is able to distinguish healthy from pre-breakdown events and therefore predicts breakdowns. Fourthly, a sensitivity analysis on the trained neural network suggests that a pressure signal is an indication for a breakdown

formation in the XBox-2 accelerator. Experiments are carried out in order to validate this hypothesis on the succeeding XBox-3 accelerating unit at CERN.

The XBox-2 data was not recorded with a machine learning application in mind. An improved way to store data could significantly increase productivity and might even lead to new findings. Limited time was allocated to validate this hypothesis, but further experiments will be carried out in the future to confirm the correctness of the identified precursors.

To summarize, this thesis analyzes the basic operating principles of linear and circular accelerators and thereby motivates accelerators and explains CERN's high-gradient test stands. Operational data pertaining to a prototype linear particle accelerator is analyzed using machine learning methods. A framework for data analysis is created, optimized and applied on data of an experimental linear accelerating structure. A hypothesis for breakdown occurrence is extracted from a trained neural network and tested in the lab. Further research is required before machine learning models can be used to derive a breakdown detection and recovery algorithm in RF cavities fully based on machine learning.

References

[12] "Radiofrequency cavities". In: (Sept. 1, 2012). URL: https://cds.cern.ch/record/1997424.

[22] *The ATLAS Open Data 13 TeV docs*. CERN. Jan. 21, 2022. URL: http://opendata.atlas.cern/release/2020/documentation/index.html (visited on 01/21/2022).

[Aic+18] Markus Aicheler et al. *The Compact Linear Collider (CLIC) – Project Implementation Plan*. CERN, Feb. 1, 2018. DOI: https://doi.org/10.23731/CYRM-2018-004.

[BC] Nils Braun and Maximilian Christ. *tsfresh*. URL: https://tsfresh.com/ (visited on 02/01/2022).

[Bur+18] Philip N. Burrows et al. *The Compact Linear Collider (CLIC) – 2018 Summary Report*. CERN, Feb. 1, 2018. DOI: https://doi.org/10.23731/CYRM-2018-002.

[Col+20] European Strategy Group Collaboration et al. "Update of the European Strategy for Particle Physics". In: *CERN Council, Geneva* (2020). URL: https://cds.cern.ch/record/2721370 (visited on 11/11/2021).

[Col+21] Andrew Collette et al. *HDF5 for python*. Aug. 4, 2021. URL: https://docs.h5py.org/en/stable/index.html (visited on 10/06/2021).

[Ein+05] Albert Einstein et al. "On the electrodynamics of moving bodies". In: *Annalen der physik* 17.10 (1905), pp. 891–921.

[Fis20] Lorenz Fischl. "FEM Triangulation for the Poisson Problem on Polygonal Domains with Neural Networks". Bachelor Thesis. 2020.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2016. URL: http://www.deeplearningbook.org.

[Gun13] Helmut Gunther. *Die Spezielle Relativitätstheorie: Einsteins Welt in einer neuen Axiomatik*. 1st ed. Springer Fachmedien Wiesbaden, 2013. ISBN: 3658007125.

[Isi24] Gustav Ising. "Prinzip einer Methode zur Herstellung von Kanalstrahlen hoher Voltzahl". In: *Ark. Mat. Astron. Fys.* 18 (1924), pp. 1–4. URL: https://cds.cern.ch/record/433984.

[KA15] Alfons Kemper and Eickler André. *Datenbanksysteme: Eine Einführung*. ger. De-Gruyter-Oldenbourg-Studium. De Gruyter Oldenbourg, Sept. 25, 2015. ISBN: 978-3110443752. URL: https://lead.to/amazon/com/?op=bt&la=de&cu=usd&key=3110443759.

[Kam14] Ahmad Anwar Kamal. *Particle physics*. Graduate texts in physics. Berlin [u.a.]: Springer, 2014. ISBN: 3642386601.

[Kov10] Jan Wilhelm Kovermann. "Comparative Studies of High-Gradient Rf and Dc Breakdowns". 2010. URL: http://cds.cern.ch/record/1330346.

[LL17] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[LP96] Elliot Leader and Enrico Predazzi. *An introduction to gauge theories and modern particle physics*. Cambridge monographs on particle physics, nuclear physics, and cosmology. Cambridge [u.a.]: Cambridge Univ. Pr., 1996. ISBN: 052157742X.

[Luc+19] Thomas Geoffrey Lucas et al. "Dependency of the capture of field emitted electron on the phase velocity of a high-frequency accelerating structure". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 914 (2019), pp. 46–52. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2018.10.166. URL: https://www.sciencedirect.com/science/article/pii/S0168900218314803.

[MHM18] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[Nat21a] National Instruments Corporation. *TDMS File Format Internal Structure*. National Instruments Corporation. Aug. 1, 2021. URL: https://www.ni.com/en-us/support/documentation/supplemental/07/tdms-file-format-internal-structure.html (visited on 09/03/2021).

[Nat21b] National Instruments Corporation. *The NI TDMS File Format*. National Instruments Corporation. June 1, 2021. URL: https://www.ni.com/en-us/support/documentation/supplemental/06/the-ni-tdms-file-format.html (visited on 08/20/2021).

[Nat21c] National Instruments Corporation. *What is LabVIEW?* National Instruments Corporation. Aug. 11, 2021. URL: https://www.ni.com/en-us/shop/labview.html (visited on 09/06/2021).

[Nav17] Jorge Giner Navarro. "Breakdown studies for high-gradient rf warm technology in: clic and hadron therapy linacs". PhD thesis. Universitat de València, 2017.

[New40] Isaac Newton. *Philosophiae naturalis principia mathematica*. Vol. 2. Barrillot, 1740.

[Nob13] Nobel Prize Outreach AB 2021. "The Nobel Prize in Physics 2013". In: *The 2013 Nobel Prize in Physics - Press release* (Sept. 10, 2013). URL: https://www.nobelprize.org/prizes/physics/2013/press-release/.

[Nob84] Nobel Prize Outreach AB 2021. "The Nobel Prize in Physics 1984". In: *The 1984 Nobel Prize in Physics - Press release* (Sept. 11, 1984). URL: https://www.nobelprize.org/prizes/physics/1984/press-release/.

[Obe+22] Christoph Obermair et al. *Explainable Machine Learning for Breakdown Prediction in High Gradient RF Cavities*. Tech. rep. Feb. 2022. arXiv: 2202.05610. URL: https://cds.cern.ch/record/2803014.

[Oku89] Lev B Okun. "The concept of mass". In: *Physics today* 42.6 (1989), pp. 31–36.

[Pan21] Pandas development team. *Package overview - pandas 1.3.3 documentation*. the pandas development team. Sept. 20, 2021. URL: https://pandas.pydata.org/docs/getting_started/overview.html (visited on 09/12/2021).

[Per00] Donald H. Perkins. *Introduction to High Energy Physics*. 4th ed. Cambridge University Press, 2000. DOI: 10.1017/CBO9780511809040.

[Pyt21a] Python Software Foundation. *csv — CSV File Reading and Writing*. Python Software Foundation. Sept. 19, 2021. URL: https://docs.python.org/3/library/csv.html (visited on 09/20/2021).

[Pyt21b] Python Software Foundation. *pickle - Python object serialization*. Python Software Foundation. Sept. 19, 2021. URL: https://docs.python.org/3.9/library/pickle.html (visited on 09/20/2021).

[Ree20] Adam Reeve. *npTDMS documentation*. Aug. 24, 2020. URL: https://nptdms.readthedocs.io/en/stable/index.html (visited on 10/04/2021).

[SCE22] SCE Department. *CERN maps*. CERN. Mar. 1, 2022. URL: https://maps.web.cern.ch (visited on 03/01/2022).

[Sir+20] Albert M Sirunyan et al. "A measurement of the Higgs boson mass in the diphoton decay channel". In: *Physics Letters B* 805 (2020), p. 135425.

[Sto+18] Mike Stonebraker et al. "C-Store: A Column-Oriented DBMS". In: *Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker*. Association for Computing Machinery and Morgan; Claypool, 2018, pp. 491–518. ISBN: 9781947487192. URL: https://doi.org/10.1145/3226595.3226638.

[Stu19] Thomas Studer. *Relationale Datenbanken: Von den theoretischen Grundlagen zu Anwendungen mit PostgreSQL*. ger. Springer Berlin Heidelberg, Oct. 29, 2019. ISBN: 9783662589755.

[Tie+21] Eite Tiesinga et al. "CODATA recommended values of the fundamental physical constants: 2018". In: *Reviews of Modern Physics* 93.2 (2021), p. 025010.

[VM] Xabier Cid Vidal and Ramon Cid Manzano. *rf cavities*. CERN. URL: https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.rf_cavities (visited on 01/23/2022).

[Wid28] Rolf Widerøe. "Über ein neues Prinzip zur Herstellung hoher Spannungen". In: *Archiv für Elektrotechnik* 21.4 (1928), pp. 387–406.

[Woo16] Nicholas Woodhouse. *Spezielle Relativitätstheorie*. ger. 1st ed. Springer-Lehrbuch Masterclass. Berlin, Heidelberg: Springer Berlin Heidelberg Imprint: Springer Spektrum, 2016. ISBN: 3662463733.

[Wue+17] Walter Wuensch et al. "Statistics of vacuum breakdown in the high-gradient and low-rate regime". In: *Phys. Rev. Accel. Beams* 20 (1 Jan. 2017), p. 011007. DOI: 10.1103/PhysRevAccelBeams.20.011007. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.20.011007.

10 Appendix

10.1 Represent Tdms Files as a Database

Tabular data formats consist of columns and rows whereas tables in databases consist of tuples and attributes. A tuple is one entry of a database. A set of tuples that all have the same structure is called a table. An attribute of a tuple is a single value while an attribute of a table is referred to as all values of a single attribute of all tuples. A collection of tables is called a database.

Definition 10.1 (tuple, table, attribute, database)**.** *Let $n \in \mathbb{N}$ and $D_1, \ldots, D_n$ be sets, called domains and let $A = \{a_1, \ldots, a_n\}$ be a set of names. Then a tuple is a function mapping A to the union of the domains that satisfies*

$$t : A \to \bigcup_{i \in 1,\ldots,n} D_i$$
$$\forall i \in 1, \ldots, n : a_i \mapsto d_i \in D_i.$$

A set of tuples that all have the same set of names and domains is a table. The projection of a table $T = \{t_i : i \in I\}$ to its k-th component is defined as the multi-set $\pi_k(T) = \{t_i(a_k) : i \in I\}$ and is called an attribute (or column) of the table. A set of tables is a database.

In the field of databases values are called by their column names (attributes) while in tabular data formats they are called by numbers (indices). The main difference to other tabular data formats, however, is the connectivity between tables.

In a DBMS the domains are defined by the data types of each attribute. These help to reserve the required amount of space when storing entries. Typical DBMS contain the data types numbers, dates and strings. Numeric data types typically have different categories like integer, float and identification numbers called *id*. In specialized applications a so called blob (binary large object) can be used to store any data in a binary format. Arrays are typically saved as blobs if the DBMS can handle them at all [KA15, Chapter 4].

One of the main advantages of databases is the possibility to store dependencies from one tuple to another. Setting two tuples in relation to each other requires an identification. A table is defined as a set of tuples. Tuples can not necessarily be uniquely identified by the image of a single name. However, it is of great interest to know the minimal set of attribute names of a table that uniquely identifies all tuples of that table. A set of attribute names with that property is called a primary key. Dependencies of tuples with each other is implemented with keys [Stu19, Chapter 2].

Definition 10.2 (primary key, foreign key)**.** *Let T be a table with attribute names A. Any subset K of the attribute name set A is called primary key of T iff*

$$\forall t, t' \in T : [\forall a \in K : t(a) = t'(a)] \Longrightarrow t = t'.$$

Let T' be a table with attribute names A'. A foreign key is a subset K' of A' that also is a primary key of T.

Data reconfiguration like joining tables are summarized as relational algebra. These operations use primary and foreign keys for merging data. Note that a foreign key can be a primary key of its own table and thus represents a self reference.

The table names, attribute names and data types are typically summarized in a table called relational scheme.

Definition 10.3 (relation scheme)**.** *Let $k \in \mathbb{N}$ be the number of tables and $n_k \in \mathbb{N}$ be the number of attributes per table. Let the table names be given by $t_1, \ldots, t_k$ and the attribute names be given by $A^k = \{a_1^k, \ldots, a_{n_k}^k\}$ where the first $\{a_1^k, \ldots a_{l_k}^k\}$ names form a primary key of the table t_k. Let the domains be given by their data types $D_k = \{d_1^k, \ldots, d_n^k\}$, then the relation scheme is given by*

$$\begin{aligned} t_1 :&\{[\underline{a_1^1 : d_1^1, \ldots, a_{l_1}^1 : d_{l_1}^1}, \ldots, a_{n_1}^1 : d_{n_1}^1]\}, \\ &\vdots \\ t_k :&\{[\underline{a_1^k : d_1^k, \ldots, a_{l_k}^1 : d_{l_k}^1}, \ldots, a_{n_k}^k : d_{n_k}^k]\}. \end{aligned}$$

The data of the XBox-2 test stand is stored in tdms files. The ease of usability in Python, the first requirement for the framework's data type, is tested by creating a database that could store tdms files. Initially a database on the conceptual level is created with a procedure presented by Alfons Kemper [KA15, Chapter 2].

Each *channel* has exactly one parent *group* while each *group* has exactly one parent *file*. The relationship that each child has exactly one parent and each parent can have multiple children is called $(1 : N)$. These three entities all have the attribute *name*. The main data is stored with the attribute *data* of the channel entity. The *name* and *value* of the properties are attributes of the *property* entity which is shared between *files*, *groups*, and *channels*. Thus the three entities all have the relationship *has* with the same entity *property*. Each *file*, *group*, and *channel* can have multiple properties and each property can be assigned to multiple entities. This relationship is called $(N : M)$.

In Figure 17 the ER (entity relationship) diagram of tdms files is displayed. The *file*, *group*, and *channel* entities are depicted on the right hand side. The shared

properties are placed on the left hand side of the figure. In an ER diagram each entity is represented by a rectangle, each connection between entities by a diamond, and each attribute by an ellipse. The connection types $(1 : N)$ and $(N : M)$ are marked at the connecting lines. Note that the attribute *data* of *channel* contains arrays which are not available in every DBMS [KA15, Chapter 2].

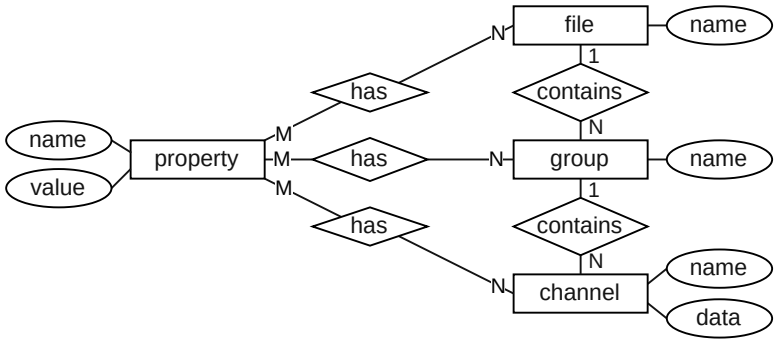


Figure 17: Entity relationship diagram of tdms files

The entity relationship model in Figure 17 can be transferred to a UML (Unified Modeling Language) diagram. This diagram consists of tables of the database. Each table has a separate box. The table name is located in the first row and its attributes in the following rows. The primary and foreign keys are marked with the abbreviations PK and FK. There is a table in the UML diagram for each entity in the ER diagram. The names of files, groups and channels can be the same. Therefore, an extra attribute *id* needs to be added for every entity in order to ensure the uniqueness of each tuple. Ids are unique consecutive numbers that identify each tuple of a table. There is a table for each relationship that contains two foreign keys, one for each related attribute. A visualization of the UML diagram with connecting lines between foreign and primary keys can be seen in Figure 18 [KA15, Chapter 3].

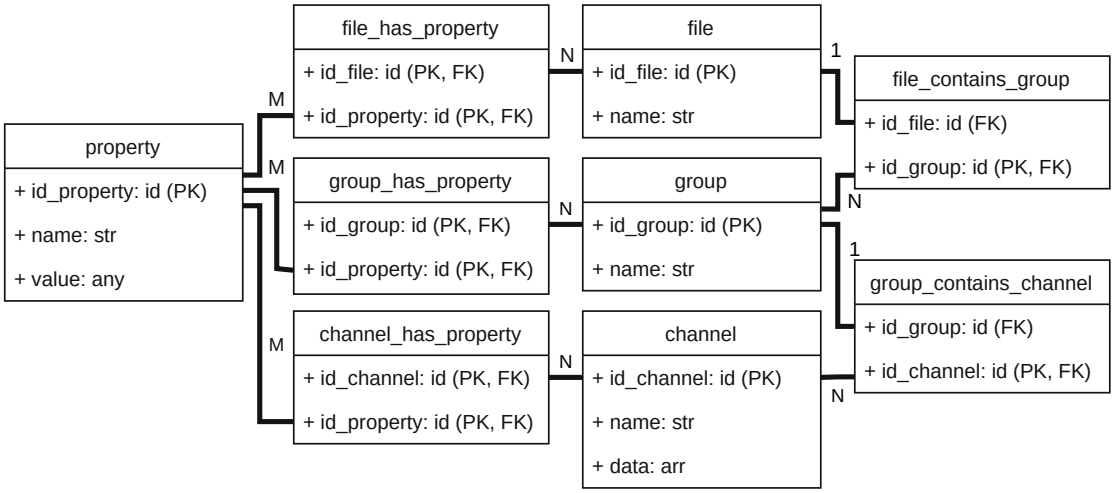


Figure 18: Conceptual model of tdms files as a UML diagram

The tables *file* and *file_contains_group* both have the same primary key, *id_file*, because of the $(1 : N)$ relationship between *file* and *group*. Tables that have the same primary keys can be merged into a single table. Similarly, *group* and *group_contains_channel* can be merged into a single table. Consequently, we remove the two *contain* relation tables and add the attribute *contained_in_file_id* to *group* and *contained_in_group_id* to *channel*. A more compact version of the UML diagram can be created by the union of these tables. The primary keys of *group_has_property* and *channel_has_property* cannot be included in another table because of their $(N : M)$ relationship. The improved version of the conceptual model of tmds files can be seen as a relation scheme in table 4.

file	:	{[id_file: id, name: str]}
group	:	{[id_group: id, name: str, contained_in_file_id: id]}
channel	:	{[id_channel: id, name: str, data: arr, contained_in_group_id: id]}
property	:	{[id_property: id, name: str, value: any]}
file_has_property	:	{[id_file: id, id_property: id]}
group_has_property	:	{[id_group: id, id_property: id]}
channel_has_property	:	{[id_channel: id, id_property: id]}

Table 4: Improved conceptual model of tdms files in a relational scheme

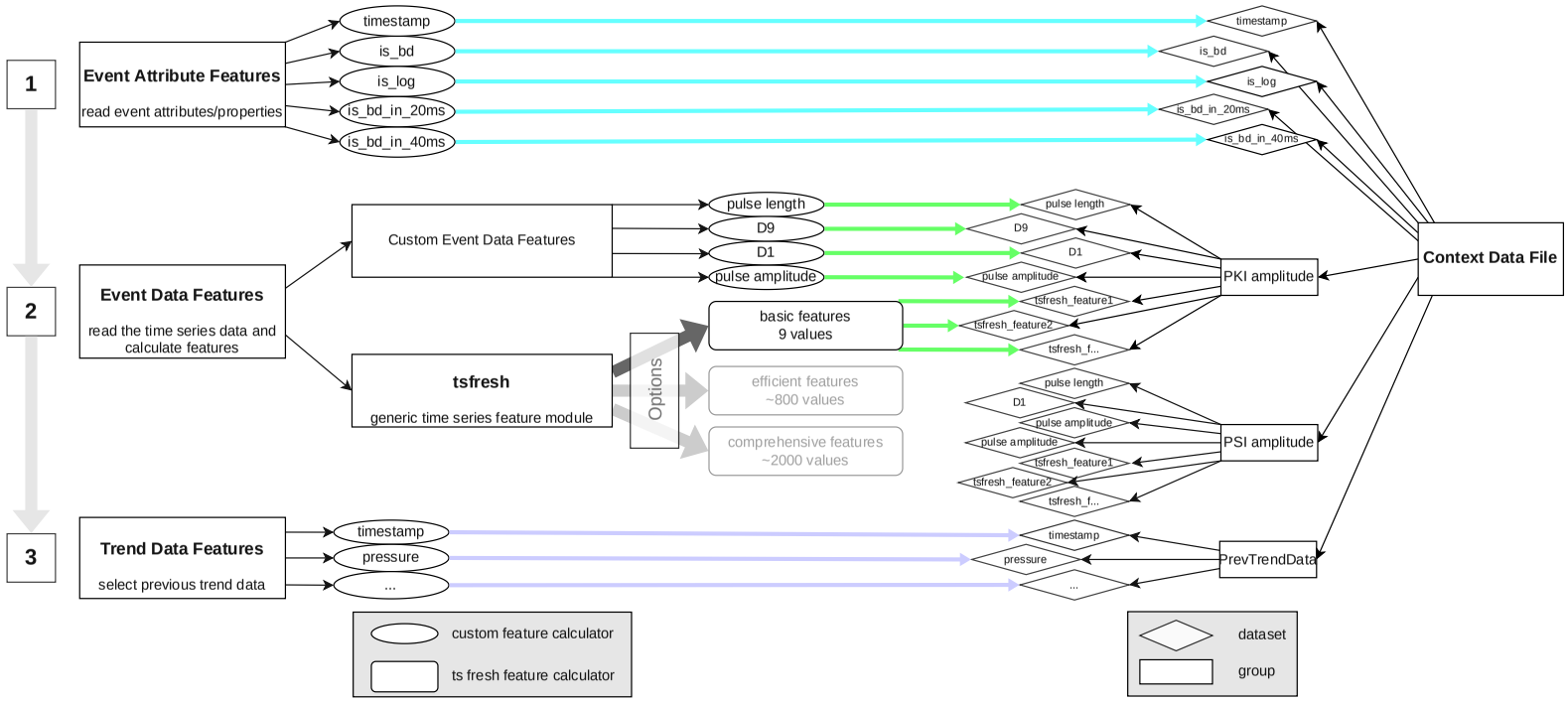
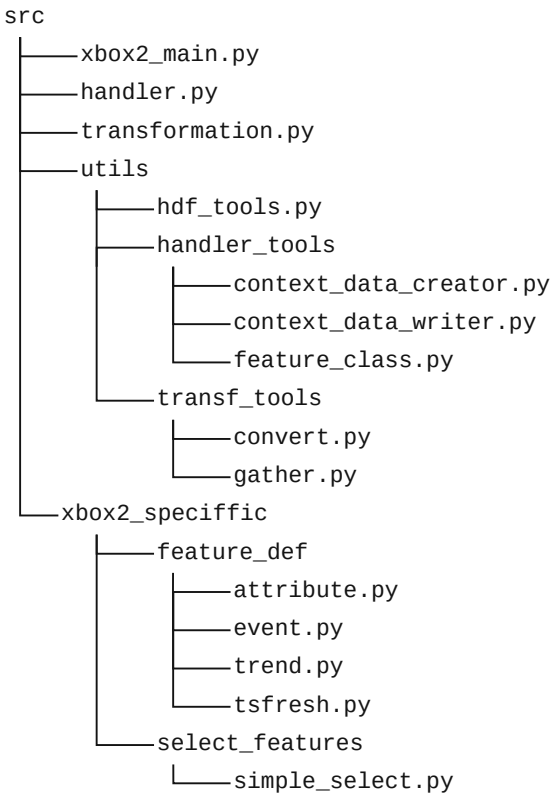


Figure 13: Model of the feature calculators on the left and their locations in the context data file on the right

10.2 Implementation

Figure 19: Directory structure of the implementations



Listing 1: Implementation of xbox2_main.py

```
"""
This is the main program for the analysis of the XBox-2 accelerator.
It combines all steps of the machine learning framework:
* transformation from tdms to hdf files
* data exploration with a data handler
* modeling of a neural network for breakdown prediction
"""
from pathlib import Path
from collections import namedtuple
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense
from sklearn.model_selection import StratifiedKFold
from src.transformation import transform
from src.handler import XBox2ContextDataCreator
from src.utils import hdf_tools
from src.xbox2_speciffic.select_features.simple_select import select_data


def transformation(src_dir: Path, work_dir: Path) -> None:
    """
    calls the transformation function, combines all trend data and
    re-formats timestamps
    :param src_dir: directory of the data source (where tdms files are located)
    :param work_dir: the working directory, destination of the transformation
    """
    transform(tdms_dir=src_dir,
              hdf_dir=work_dir)

    gathered_trend_data = work_dir / "TrendDataExtLinks.hdf"
    combined_trend_data_path = work_dir / "combined.hdf"

    hdf_tools.merge(source_file_path=gathered_trend_data,
                    dest_file_path=combined_trend_data_path)
    hdf_tools.convert_iso8601_to_datetime(file_path=combined_trend_data_path)
    hdf_tools.sort_by(file_path=combined_trend_data_path, sort_by_name="Timestamp")


def data_handling(work_dir: Path) -> None:
    """
    creates the context data
    :param work_dir: the working directory where the gathered data files are located
    """
    gathered_event_data_path = work_dir / "EventDataExtLinks.hdf"
    context_data_file_path = work_dir / "context.hdf"
    combined_trend_data_path = work_dir / "combined.hdf"

    creator = XBox2ContextDataCreator(ed_file_path=gathered_event_data_path,
                                      td_file_path=combined_trend_data_path,
                                      dest_file_path=context_data_file_path)
    creator.manage_features()


def get_model() -> keras.models.Model:
    """
    creates and compiles a neural network that is ready for training
    :return: a compiled keras neural network
    """
    activation_func = tf.nn.relu
    input_layer = Input(shape=191)
    layer1 = Dense(units=191, activation=activation_func)(input_layer)
    layer2 = Dense(units=191, activation=activation_func)(layer1)
    output_layer = Dense(units=1, activation=tf.nn.sigmoid)(layer2)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)
    model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Adam(),
                  metrics=[keras.metrics.AUC()])
```

```
69      return model
70
71
72  def modeling() -> np.ndarray:
73      """
74      creates test data and does a k-fold cross validation to test
75      a neural network's performance with balanced accuracy
76      :return: list of the balanced accuracies done by the k-fold cross validation
77      """
78      x, y = select_data()
79      y = y.astype(dtype=int)
80
81      results_list = []
82      k_fold = StratifiedKFold(n_splits=5, shuffle=True)
83      batch_size = 10
84      num_epochs = 20
85      data = namedtuple("data", ["x", "y"])
86      for train, test in k_fold.split(x, y):
87          train = data(x[train], y[train])
88          test = data(x[test], y[test])
89
90          model = get_model()
91          model.fit(train.x, train.y, batch_size=batch_size, validation_split=0.3,
            ↪  epochs=num_epochs,
92                    verbose=True)
93
94          predictions = np.around(model.predict(x=test.x)[:, 0]).astype(dtype=int)
95          c_m = tf.math.confusion_matrix(labels=test.y, predictions=predictions)
96          bal_acc = (c_m[0, 0] / (c_m[0, 0] + c_m[1, 0]) + c_m[1, 1] / (c_m[0, 1] + c_m[1,
            ↪  1])) / 2
97          results_list.append(bal_acc)
98
99      return np.array(results_list)
100
101
102 if __name__ == "__main__":
103     transformation(src_dir=Path("~/project_data/CLIC_DATA_Xbox2_T24PSI_2/").expanduser(),
104                    work_dir=Path("~/output_files/").expanduser())
105     data_handling(work_dir=Path("~/output_files/").expanduser())
106     print(modeling().mean())
```

Listing 2: Implementation of hdf_tools.py

```
"""
This module contains implementations of useful functions for hdf-files.
* merge: unifies all hdf-datasets in a file that have the same name;
    searches for all datasets in a file
* clean: the clean_by_row cleans "smelly" values row by row
* sort: sorts all datasets according to one of them
* conversion to df: convert_to_df converts an hdf file into a pandas dataframe
"""
from pathlib import Path
import logging
import re
import typing
import numpy as np
import h5py
import pandas as pd

logger = logging.getLogger(__name__)


def merge(source_file_path: Path, dest_file_path: Path) -> None:
    """
    merges specific parts of datasets in hdf files
    In the first layer of the hdf-file structure there have to be groups only.
    In the following layer each group has to have hdf-datasets of the same name.
    Then this function merges (concatenates) all datasets into a file that has the
    merged datasets as children of the hdf-file-group (the root).
    :param source_file_path: file of the un-merged groups with same data set names
    :param dest_file_path: file where the merged datasets will be stored at the root
    """
    with h5py.File(source_file_path, mode="r") as source_file, \
            h5py.File(dest_file_path, mode="a") as dest_file:
        first_grp = source_file.values().__iter__().__next__()
        for channel_name in first_grp.keys():  # the channel names are always the same
            logger.debug("currently merging: %s", channel_name)
            data = np.concatenate([grp[channel_name][:] for grp in source_file.values()])
            dest_file.create_dataset(name=channel_name, data=data, chunks=True)


def convert_iso8601_to_datetime(file_path: Path, also_convert_attrs: bool = True) -> None:
    """
    converts datasets and attributes of strings that contain datetime values of iso8601
    format to the numpy datetime format
    :param file_path: path of the hdf file to convert
    :param also_convert_attrs: boolean value to define if attrs datetime should
        also be converted
    """
    def convert_attrs(_: str, hdf_obj):
        """
        a visitor function (hdf.File.visititems()) converts all the attributes
        of the given hdf_obj.
        """
        for attrs_key, val in hdf_obj.attrs.items():
            try:
                val = pd.to_datetime(val.astype(str), format="%Y-%m-%dT%H:%M:%S.%f")
            except ValueError:
                pass
            else:
                val = val.to_numpy(np.datetime64)
                del hdf_obj.attrs[attrs_key]
                hdf_obj.attrs.create(name=attrs_key, data=np.array(val).
                                     astype(h5py.opaque_dtype(val.dtype)))

    with h5py.File(file_path, mode="r+") as file:
        if also_convert_attrs:
            convert_attrs("/", file)
            file.visititems(convert_attrs)

        for key, channel in list(get_all_dataset_items(file)):
```

```
            try:
                data = pd.to_datetime(data=channel[:].astype(str),
                                      format="%Y-%m-%dT%H:%M:%S.%f")
            except ValueError:
                pass
            else:
                data = data.to_numpy(np.datetime64)
                del file[key]
                file.create_dataset(name=key,
                                    data=data.astype(h5py.opaque_dtype(data.dtype)))


def _check_corruptness(arr: np.ndarray):
    """
    checks if the input array contains a healthy or smelly (inf or nan) value
    :param arr: input array to check
    :return: array with boolean values
        True if the value at that index was healthy; False if it was corrupt
    """
    if np.issubdtype(arr.dtype, np.number):
        is_corrupt = np.isnan(arr) | np.isinf(arr)
    elif np.issubdtype(arr.dtype, np.datetime64):
        is_corrupt = np.isnat(arr)
    else:
        raise NotImplementedError("Corrupt data is only known for "
                                  "numeric and datetime values.")
    return is_corrupt


def clean_by_row(file_path: Path) -> None:
    """
    removes smelly rows
    column = dataset of the hdf-file
    row = values with the same index from all columns
    :param file_path: the path of the hdf-file containing datasets of same length
    """
    with h5py.File(file_path, "r+") as file:
        shape = file.values().__iter__().__next__().shape  # shape of the first dataset
        is_corrupt = np.zeros(shape, dtype=bool)
        for channel in file.values():
            is_corrupt |= _check_corruptness(channel[:])
        new_shape = (sum(~is_corrupt),)
        for channel in file.values():
            logger.debug("cleaning channel: %s", channel)
            data = channel[~is_corrupt]
            channel.resize(size=new_shape)
            channel[...] = data


def sort_by(file_path: Path, sort_by_name: str) -> None:
    """
    sorts all datasets with respect to one specific dataset (given by the key)
    this is done inplace, without copying data
    :param file_path: path of the hdf-file containing datasets of the same length
    :param sort_by_name: name of the dataset to be sorted
    """
    with h5py.File(file_path, "r+") as file:
        indices_order = file[sort_by_name][:].argsort()
        for channel in get_all_dataset_values(file):
            data = channel[:]
            channel[...] = data[indices_order]


def hdf_path_combine(*argv: str) -> str:
    """
    concatenates hdf path with "/" in between
    works similar to Path(str, str, str) or the / operator for Path objects but for
    hdf-paths (so with string objects)
    :param argv: the group names for concatenation
    :return: the concatenated path string
    """
    path = "/" + "/".join(argv)
```

```
    path = re.sub('//+', '/', path)
    return path


def _get_all_dataset_items_rec(
        hdf_path: str, hdf_obj: typing.Union[h5py.File, h5py.Dataset, h5py.Group])\
            -> typing.Generator:
    """
    a private generator that returns all items that are children of the value hdf object
    :param hdf_path: hdf-object
    :return: generator that returns all children hdf-dataset items of that hdf-object
    """
    if isinstance(hdf_obj, h5py.Dataset):
        yield hdf_path, hdf_obj
    else:
        for key, val in hdf_obj.items():
            yield from ((hdf_path_combine(hdf_path, key), val)
                        for key, val in _get_all_dataset_items_rec(key, val))


def get_all_dataset_items(hdf_obj, path: str = "/") -> typing.Generator:
    """
    a generator that returns all items that are children of the value hdf object
    :param hdf_obj: starting hdf_obj (file, group, dataset)
    :param path: the path of the starting hdf_obj
    :return: generator that returns all children hdf-dataset items of the given path
    """
    yield from _get_all_dataset_items_rec(path, hdf_obj)


def get_all_dataset_values(value: typing.Union[h5py.File, h5py.Dataset, h5py.Group]) -> \
        typing.Generator:
    """
    a generator that returns all values that are children of the value hdf object
    :param value: the value to recursively go through all values
    :return: generator that returns all children hdf-dataset values of that hdf-object
    """
    if isinstance(value, h5py.Dataset):
        yield value
    else:
        for val in value.values():
            yield from get_all_dataset_values(val)


def hdf_to_df(file_path: Path, hdf_path: str = "/") -> pd.DataFrame:
    """
    converts hdf files into pandas dataframes
    :param file_path: file path of the hdf-file to convert
    :param hdf_path: inner hdf-path whose children datasets should be converted
    :return: pandas dataframe
    """
    with h5py.File(file_path, "r") as file:
        return pd.DataFrame(data={path[1:].replace("/", "__").replace(" ", "_"): val[:]
                                  for path, val in get_all_dataset_items(file[hdf_path],
                                                                         hdf_path)})


def hdf_to_df_selection(file_path: Path, selection, hdf_path: str = "/") -> pd.DataFrame:
    """
    converts selection of hdf files into pandas dataframes
    :param file_path: file path of the hdf-file to convert
    :param hdf_path: inner hdf-path whose children datasets should be converted
    :return: pandas dataframe
    """
    with h5py.File(file_path, "r") as file:
        return pd.DataFrame(
            data={path[1:].replace("/", "__").replace(" ", "_"): val[selection]
                  for path, val in get_all_dataset_items(file[hdf_path], hdf_path)})
```

10.2.1 Transformation

Listing 3: Implementation of transformation.py

```python
"""This module applies conversion and gathering on XBox-2 data."""
from pathlib import Path
import logging
import h5py
import numpy as np
from src.utils.transf_tools.convert import Convert
from src.utils.transf_tools.gather import Gatherer

logger = logging.getLogger(__name__)


def transform(tdms_dir: Path, hdf_dir: Path) -> None:
    """
    transforms all tdms files to hdf files, filters faulty data and gathers hdf
    groups with depth 1 of the hdf files into one hdf file with external links
    :param tdms_dir: input directory with tdms files
    :param hdf_dir: output directory with hdf files
    """
    Path(hdf_dir, "data").mkdir(parents=False, exist_ok=True)

    # read tdms files, convert them to hdf5 and write them into hdf_dir/data/
    Convert(check_already_converted=True, num_processes=4) \
        .from_tdms(tdms_dir) \
        .to_hdf(hdf_dir / "data").run()

    # combine all events and trend data sets into one hdf5 file with external links
    # if the time series segments are healthy

    def td_func_to_fulfill(file_path: Path, hdf_path: str) -> bool:
        with h5py.File(file_path, "r") as file:
            grp = file[hdf_path]
            ch_shapes = [ch.shape[0] for ch in grp.values()]
            len_equal = all(ch_shape == ch_shapes[0] for ch_shape in ch_shapes)
            num_of_samples = 35
            return len_equal and len(ch_shapes) == num_of_samples

    Gatherer(if_fulfills=td_func_to_fulfill, on_error=False, num_processes=4)\
        .gather(src_file_paths=hdf_dir.glob("data/Trend*.hdf"),
                dest_file_path=hdf_dir / "TrendDataExtLinks.hdf")

    def ed_func_to_fulfill(file_path: Path, hdf_path: str) -> bool:
        with h5py.File(file_path, "r") as file:
            grp = file[hdf_path]
            ch_len = [ch.shape[0] for ch in grp.values()]

            acquisition_window = 2e-6  # time period of one event is 2 microseconds

            # acquisition card NI-5772 from National Instruments
            # see https://www.ni.com/en-us/support/model.ni-5772.html
            sampling_frequency_ni5772 = 1.6e9
            num_of_values_ni5772 = acquisition_window * sampling_frequency_ni5772
            number_of_signals_monitored_with_ni5772 = 8

            # acquisition card NI-5761 from National Instruments
            # see https://www.ni.com/en-us/support/model.ni-5761.html
            sampling_frequency_ni5761 = 2.5e8
            num_of_values_ni5761 = acquisition_window * sampling_frequency_ni5761
            number_of_signals_monitored_with_ni5761 = 8

            def has_smelly_values(data) -> bool:
                return any(np.isnan(data) | np.isinf(data))

            return grp.attrs.get("Timestamp", None) is not None \
                and ch_len.count(num_of_values_ni5772) == \
                   number_of_signals_monitored_with_ni5772 \
                and ch_len.count(num_of_values_ni5761) == \
                   number_of_signals_monitored_with_ni5761 \
                and not any(has_smelly_values(ch[:]) for ch in grp.values())
```

```
70      Gatherer(if_fulfills=ed_func_to_fulfill, on_error=False, num_processes=1)\
71          .gather(src_file_paths=hdf_dir.glob("data/EventData_201804*.hdf"),
72                  dest_file_path=hdf_dir / "EventDataExtLinks.hdf")
```

Listing 4: Implementation of convert.py

```python
"""This module contains classes that read data and convert it to a different
file format."""
import os
from time import time
from pathlib import Path
import multiprocessing as mp
import logging
from functools import partial
import nptdms
import h5py

logger = logging.getLogger(__name__)


def _convert_file(tdms_file_path: Path, hdf_dir: Path) -> None:
    """
    converts one tdms file to an hdf file with the nptdms as_hdf function
    :param tdms_file_path: file path of the tdms file
    :param hdf_dir: file path of the hdf file (destination)
    """
    t_0 = time()
    with nptdms.TdmsFile(tdms_file_path) as tdms_file:
        logger.debug("reading tdms file  %40s     took: %10.10s sec",
                     tdms_file_path.stem, time() - t_0)
        hdf_file_path = hdf_dir / tdms_file_path.with_suffix(".hdf").name
        t_0 = time()
        tdms_file.as_hdf(hdf_file_path, mode="w", group="/")
        logger.debug("tdms2hdf + writing %40s     took: %10.10s sec",
                     tdms_file_path.stem, time() - t_0)


class Convert:  # pylint: disable=too-few-public-methods
    """very general converter object, that can be adapted to other input and output
     data formats"""

    def __init__(self,
                 check_already_converted: bool = True,
                 num_processes: int = 2):
        """
        initializes the converter
        :param check_already_converted:  It is possible that part of the files has already
            been converted. check_already_converted defines whether only the files
            that have not been converted yet or all files will be converted
        :param num_processes: number of processes for parallel conversion
        """
        self.check_already_converted = check_already_converted
        self.num_processes = num_processes

    def from_tdms(self, tdms_dir: Path):
        """
        adds the source directory where the tdms files are located and returns
        a ConvertFromTdms object
        :param tdms_dir: file path of the directory where the tdms files are located
        :return: a ConvertFromTdms object
        """
        return ConvertFromTdms(tdms_dir,
                               self.check_already_converted, self.num_processes)


class ConvertFromTdms(Convert):
    """adds the from_directory (source) for the converter"""

    def __init__(self, tdms_dir: Path, check_already_converted: bool, num_processes: int):
        """
        initializes the ConvertFromTdms class object, inherits from Convert.
        tdms_dir is the source directory where the tdms files ready for conversion
        are located.
        :param tdms_dir: source directory of tdms files
```

```
        :param check_already_converted: only convert the not yet converted files
        :param num_processes: number of processes for parallel conversion
        """
        super().__init__(check_already_converted, num_processes)
        self.tdms_dir = tdms_dir

    def to_hdf(self, hdf_dir: Path):
        """
        adds the destination directory where the hdf files will be stored and returns
        a ConvertFromTdmsToHdf object
        :param hdf_dir: path of the directory where the hdf-files will be stored
        :return: a ConvertFromTdmsToHdf object
        """
        return ConvertFromTdmsToHdf(
            hdf_dir, self.tdms_dir, self.check_already_converted, self.num_processes)


class ConvertFromTdmsToHdf(ConvertFromTdms):
    """adds the destination directory (destination) for ConvertFromTdms"""

    def __init__(self, hdf_dir: Path, tdms_dir: Path,
                 check_already_converted: bool, num_processes: int):
        """
        initializes the ConvertFromTdmsToHdf class object
        :param hdf_dir: destination directory of the hdf files
        :param tdms_dir: ConverterFromTdms class object
        """
        super().__init__(tdms_dir, check_already_converted, num_processes)
        self.hdf_dir = hdf_dir

    def get_tdms_file_paths_to_convert(self) -> set:
        """
        creates a set of tdms files that will be converted
        if check_already_converted -> returns the file paths that are not converted yet
        else -> return all tdms files in the tdms_dir
        :return: set of file paths that will be converted
        """
        tdms_file_paths = self.tdms_dir.glob("*.tdms")
        if self.check_already_converted:
            for path in self.hdf_dir.glob("*.hdf"):
                try:
                    # if the writing process of an hdf file was aborted prematurely,
                    # the file cannot be opened and therefore needs to be recompiled
                    h5py.File(path, "r").close()
                except OSError:
                    os.remove(path)
            hdf_file_names = set(path.stem for path in self.hdf_dir.glob("*.hdf"))
            ret = set(p for p in tdms_file_paths if p.stem not in hdf_file_names)
        else:
            ret = set(tdms_file_paths)
        if len(ret) != 0:
            logger.debug("Files to convert: %s", len(ret))
        return ret

    def run(self) -> None:
        """starts the conversion process"""
        t_tot = time()
        if self.num_processes == 1:
            for path in self.get_tdms_file_paths_to_convert():
                _convert_file(path, self.hdf_dir)
        else:
            convert_func = partial(_convert_file, hdf_dir=self.hdf_dir)
            with mp.Pool(self.num_processes) as pool:
                pool.map(convert_func, self.get_tdms_file_paths_to_convert())
        logger.debug("In total tdms to hdf5 conversion took: %s sec", time() - t_tot)
```

Listing 5: Implementation of gather.py

```python
"""The gather module contains classes that combine/group/merge hdf-groups
into one file when they are scattered in multiple files with external links."""
from typing import Callable
import logging
from pathlib import Path
from functools import partial
from collections.abc import Iterable
import h5py
from src.utils.hdf_tools import hdf_path_combine

logger = logging.getLogger(__name__)


def _get_ext_link_rec(file_path: Path, hdf_path: str, depth_to_go: int,
                      func_to_fulfill: Callable[[Path, str], bool]) -> set:
    """
    recursive function to return external links of hdf files in variable depth
    :param file_path: path of the hdf file
    :param hdf_path: hdf_path of the hdf object inside the hdf file
    :param depth_to_go: goal depth to go inside the hdf-file
    :param func_to_fulfill: the function that has to be fulfilled in order to be gathered
    :return: set of external hdf links
    """
    if depth_to_go == 0:
        return {h5py.ExternalLink(file_path, hdf_path)} if \
            func_to_fulfill(file_path, hdf_path) else set()

    if depth_to_go > 0:
        with h5py.File(file_path, "r") as file:
            children_set = set()
            for key in file[hdf_path].keys():
                children_set.update(set(_get_ext_link_rec(
                    file_path=file_path, hdf_path=hdf_path_combine(hdf_path, key),
                    depth_to_go=depth_to_go - 1, func_to_fulfill=func_to_fulfill)))
            return children_set

    raise ValueError("depth_to_go should be a non negative integer")


def _hdf_write_ext_links(source_file_path: Path, dest_file_path: Path, depth: int,
                         func_to_fulfill: Callable[[Path, str], bool]) -> None:
    """
    writes external links from the source file of given depth into the destination file
    :param source_file_path: file path of the hdf file
    :param dest_file_path: file path of the destination hdf file
    :param lock: lock to gain writing access to the dest hdf file
    :param depth: the depth in which external links will be created
    :param func_to_fulfill: the function that has to be fulfilled in order to be gathered
    """
    ext_link_list = _get_ext_link_rec(file_path=source_file_path, hdf_path="/",
                                      depth_to_go=depth, func_to_fulfill=func_to_fulfill)
    with h5py.File(dest_file_path, "a") as dest_file:
        for link in ext_link_list:
            grp_name = source_file_path.stem + link.path.replace("/", "-")
            dest_file[grp_name] = link


def _get_func_to_fulfill(on_error: bool,
                         func_to_fulfill: Callable[[Path, str], bool] = None) -> \
        Callable[[Path, str], bool]:
    """
    creates the function_to_fulfill with error handling
    if an error occurs during evaluation of the func_to_fulfill the error will be handled
    :param func_to_fulfill: filtering function
    :param on_error: boolean value that will be returned when the func_to_fulfill throws
        an error
    """
    def func_to_fulfill_with_error_handling(file_path: Path, hdf_path: str) -> bool:
```

```
        ret = on_error
        try:
            if func_to_fulfill is None:
                ret = True
            else:
                ret = func_to_fulfill(file_path, hdf_path)
        except (ValueError, SystemError, ArithmeticError,
                AttributeError, LookupError, RuntimeError):
            logger.info("Caught error for function_to_fulfill on input "
                        "(%s, %s). Returned on_error=%s", file_path, hdf_path, on_error)
        return ret

    return func_to_fulfill_with_error_handling


class Gatherer:  # pylint: disable=too-few-public-methods
    """
    gathers hdf-groups of many hdf files into one by creating external links that point
    to the original files
    therefore, data can be accessed through one file without copying it into a large file
    """

    def __init__(self, if_fulfills: Callable[[Path, str], bool] = None,
                 on_error: bool = False,
                 depth: int = 1,
                 num_processes: int = 2):
        """
        initializes the Gatherer
        :param if_fulfills: function that needs to be fulfilled so that the
            hdf-object is added to the destination file via external links
        :param on_error: boolean value that will be returned when the func_to_fulfill
             throws an error
        :param depth: the hdf-file-depth in which external links will be created
        :param num_processes: number of processors used for parallel gathering
        """
        self.func_to_fulfill = _get_func_to_fulfill(on_error, if_fulfills)
        self.depth = depth
        self.num_processes = num_processes

    def gather(self, src_file_paths: Iterable,
               dest_file_path: Path) -> None:
        """
        starts the gathering process
        :param src_file_paths: Iterable of Path objects of the source hdf-file-paths
        :param dest_file_path: Path of the destination hdf file
        """
        h5py.File(dest_file_path, "w").close()  # overwrite destination file
        multi_proc_func = partial(_hdf_write_ext_links,
                                  dest_file_path=dest_file_path,
                                  depth=self.depth,
                                  func_to_fulfill=self.func_to_fulfill)
        for path in src_file_paths:
            multi_proc_func(path)
        logger.debug("finished gathering %s", dest_file_path)
```

10.2.2 Exploration

Listing 6: Implementation of handler.py

```python
"""This module contains the class that is capable of creating the context data."""
import logging
from pathlib import Path
from dataclasses import dataclass, field
import itertools
import typing
import numpy as np
import h5py
from tqdm import tqdm
import tsfresh
from src.utils.hdf_tools import hdf_path_combine, sort_by
from src.utils.handler_tools.context_data_creator import ContextDataCreator
from src.xbox2_specifiic.feature_def.attribute import get_event_attribute_features
from src.xbox2_specifiic.feature_def.event import get_event_data_features
from src.xbox2_specifiic.feature_def.trend import get_trend_data_features
from src.xbox2_specifiic.feature_def.tsfresh import get_tsfresh
from src.utils.handler_tools.context_data_writer import \
    ColumnWiseContextDataWriter, RowWiseContextDataWriter

logger = logging.getLogger(__name__)


@dataclass
class XBox2ContextDataCreator(ContextDataCreator):
    """
    Class that manages the creation of the context data file. It contains a fixed number
    of features for each event.
    """
    ed_file_path: Path
    td_file_path: Path
    num_events: int = field(init=False) # number of events

    def __post_init__(self):
        with h5py.File(self.ed_file_path, "r") as file:
            self.num_events: int = len(file)

    def manage_features(self):
        """
        manages all feature calculation processes
        the feature list creation is done in another python file
        """
        feature_list = list(get_event_attribute_features(length=self.num_events))
        self.manage_event_attribute_features(feature_list)
        sort_by(self.dest_file_path, "Timestamp")

        feature_list = list(get_trend_data_features(
            length=self.num_events, trend_data_file_path=self.td_file_path))
        self.manage_trend_data_features(feature_list)

        feature_list = list(get_event_data_features(length=self.num_events))
        self.manage_event_data_and_tsfresh_features(event_data_features=feature_list)

        self.feature_post_processing()

    def manage_event_attribute_features(self, features: typing.List) -> None:
        """
        manages the reading and writing of attributes of the event data
        :param features: a list of EventAttributeFeatures
        """
        # calculate features
        with h5py.File(self.ed_file_path, "r") as file:
            attrs_gen = (grp.attrs for grp in file.values())
            for attrs, index in zip(attrs_gen, itertools.count(0)):
                for feature in features:
                    feature.calculate_single(index, attrs)
        # write features into the context data
        column_wise_handler = ColumnWiseContextDataWriter(self.dest_file_path,
                                                          length=self.num_events)
```

```
        for feature in features:
            column_wise_handler.write_column(feature)

    def manage_trend_data_features(self, features: typing.List) -> None:
        """
        manages the reading and writing of the trend data features
        the main concern is to find the first preceding trend data record of every event
        :param features: a list of TrendDataFeatures
        """
        # calculate features
        with h5py.File(self.td_file_path, "r") as trend_data_file, \
                h5py.File(self.dest_file_path, "r") as context_data_file:
            trend_ts = np.array(trend_data_file["Timestamp"][:])
            event_ts = np.array(context_data_file["Timestamp"][:])
            loc = np.searchsorted(trend_ts, event_ts) - 1
        cw_handler = ColumnWiseContextDataWriter(
            self.dest_file_path, length=self.num_events)
        # write features into the context data
        for feature in features:
            feature.vec = feature.calculate_all(loc)
            cw_handler.write_column(feature)

    def manage_event_data_and_tsfresh_features(self, event_data_features: typing.List) \
            -> None:
        """
        manages the calculation and writing of event data features
        event data = event data time series arrays
        custom and tsfresh features are used
        :param event_data_features: a list of EventDataFeatures
        """
        # calculate features and write tsfresh features
        row_wise_handler = RowWiseContextDataWriter(
            self.dest_file_path, length=self.num_events)
        with h5py.File(self.ed_file_path, "r") as file:
            data_gen = ({key: channel[:] for key, channel in grp.items()}
                        for grp in file.values())
            for data, index in tqdm(zip(data_gen, itertools.count(0))):
                for feature in event_data_features:
                    feature.calculate_single(index, data)

                tsfresh_df = get_tsfresh(
                    data=data, settings=tsfresh.feature_extraction.MinimalFCParameters())
                val_gen = ((hdf_path_combine(str(clm_id), str(row_id)), val)
                           for row_id, row in tsfresh_df.iterrows()
                           for clm_id, val in row.items())
                # write ts fresh features row by row
                row_wise_handler.write_row_from_external(index=index, data_iter=val_gen)
        # write custom features into the context data
        column_wise_handler = ColumnWiseContextDataWriter(
            self.dest_file_path, length=self.num_events)
        for feature in event_data_features:
            column_wise_handler.write_column(feature)

    def feature_post_processing(self):
        """
        after all features have been calculated, additional features are calculated
        the features require the information of more than just one event
        """
        with h5py.File(self.dest_file_path, "r+") as file:
            clic_label = file["/clic_label"]
            is_bd = clic_label["is_bd_in_40ms"][:-2] & \
                    clic_label["is_bd_in_20ms"][1:-1] & \
                    clic_label["is_bd"][2:]
            length = clic_label["is_bd"][:].shape

            file.require_dataset(name="is_bd", shape=length, dtype=bool,
                                 chunks=True)[:] = np.append([False, False], is_bd)
            file.require_dataset(name="is_bd_in_20ms", shape=length, dtype=bool,
                                 chunks=True)[:] = np.append([False],
                                                            np.append(is_bd, [False]))
            file.require_dataset(name="is_bd_in_40ms", shape=length, dtype=bool,
                                 chunks=True)[:] = np.append(is_bd, [False, False])
```

```
141             file.require_dataset(name="is_healthy", shape=length, dtype=bool,
142                                  chunks=True)[:] = file["clic_label/is_healthy"][:]
```

Listing 7: Implementation of feature_class.py

```python
"""This module contains a class structure of machine learning features."""
from dataclasses import dataclass, field
import typing
import logging
import h5py
import numpy as np
from src.utils.hdf_tools import hdf_path_combine

logger = logging.getLogger(__name__)


@dataclass
class CustomFeature:
    """
    a feature is a statistical property of a time series (min, max, mean, etc.) or a
    measurement of the environment that the time series was recorded
    one object of the CustomFeature class exists for every feature, it manages the
    reading, calculation and temporary storage of values
    """
    name: str  # name of the feature, also the name of the dataset in the full_hdf_path
    func: typing.Callable  # the feature function
    hdf_path: str  # path of the hdf-group where the hdf-dataset with self.name goes to
    info: str  # additional information that is stored in the hdf-group attributes

    @property
    def full_hdf_path(self) -> str:
        """
        :return: full hdf path starting from root, ending with the hdf-dataset name
        """
        return hdf_path_combine(self.hdf_path, self.name)


@dataclass
class ColumnWiseFeature(CustomFeature):
    """
    parent class of all features that are temporarily stored and written in one go
    """
    length: int
    output_dtype: typing.Union[type, np.dtype]
    vec: typing.Any = field(init=False)

    def __post_init__(self):
        self.vec = np.empty(shape=(self.length,), dtype=self.output_dtype)


class EventAttributeFeature(ColumnWiseFeature):
    """features read from the event data attributes"""
    def calculate_single(self, index: int, attrs: h5py.AttributeManager):
        """
        reads the event attribute feature by applying self.func and writes it to the
        self.vec at the given index
        :param index: index of the input event and destination index of the feature
        :param attrs: an h5py.AttributeManager object
        """
        self.vec[index] = self.func(attrs)

class TrendDataFeature(ColumnWiseFeature):
    """
    features read from the trend data attributes
    """
    def calculate_all(self, selection: np.ndarray):
        """
        reads all trend data feature at once by applying self.func and returns it
        :param selection: selection of trend data records that are stored as features
        :return: numpy array of datatype self.dtype
        """
        return self.func(selection)
```

```
69
70  @dataclass
71  class EventDataFeature(ColumnWiseFeature):
72      """features calculated from the event data array (e.g. statistical properties)"""
73      working_on_dataset: str
74
75      def calculate_single(self, index: int, data: np.ndarray):
76          """
77          calculates the event attribute feature by applying self.func and writes it to
78          the self.vec at the given index
79          :param index: index of the input event and destination index of the feature
80          :param data: all data of a single event as a dictionary
81          """
82          self.vec[index] = self.func(data[self.working_on_dataset])
```

Listing 8: Implementation of context_data_creator.py

```python
"""This module contains an abstract class structure for creating a context data file.
The ContextDataCreator class should organize the creation of the context data file."""
from dataclasses import dataclass
import logging
from pathlib import Path
import abc

logger = logging.getLogger(__name__)


@dataclass
class ContextDataCreator(abc.ABC):
    """abstract class that manages the creation of the context data file"""
    dest_file_path: Path

    @abc.abstractmethod
    def manage_features(self) -> None:
        """abstract method for the feature calculation process"""
```

Listing 9: Implementation of context_data_writer.py

```python
"""This module contains the class structure of two different context data writers."""
from pathlib import Path
import typing
from dataclasses import dataclass
import h5py
from src.utils.handler_tools.feature_class import ColumnWiseFeature


@dataclass
class ContextDataWriter:
    """abstract context data handler class. Different writing methods are inherited."""
    context_data_file_path: Path
    length: int


class ColumnWiseContextDataWriter(ContextDataWriter):  # pylint:
    ↪ disable=too-few-public-methods
    """
    simplifies the filling the context data file
    column-wise = writing single feature applied on all all events at once
    """

    def write_column(self, feature: ColumnWiseFeature) -> None:
        """
        writes a column placed in feature.vec into feature.full_hdf_path
        :param feature: a ColumnWiseFeature with a filled feature.vec
        """
        with h5py.File(self.context_data_file_path, "a") as file:
            dataset = file.require_dataset(name=feature.full_hdf_path,
                                           shape=(self.length,),
                                           dtype=feature.vec.dtype,
                                           chunks=True)
            dataset[:] = feature.vec
            dataset.attrs.create(name="info", data=feature.info)


class RowWiseContextDataWriter(ContextDataWriter):
    """
    simplifies the filling the context data file
    row-wise = writes a bunch of feature applied on one individual event
    """

    def write_row_custom_features(self, index: int, data_iter: typing.Iterable):
        """
        writes a single value into many paths given by the data_iter
        :param index: the location of the event
        :param data_iter: an iterable of tuples (hdf_path, value)
        """
        with h5py.File(self.context_data_file_path, "a") as file:
            for feature, value in data_iter:
                dataset = file.get(feature.full_hdf_path, None)
                if dataset is None:
                    dataset = file.require_dataset(
                        name=feature.full_hdf_path, shape=(self.length,),
                        dtype=type(value), chunks=True)
                    dataset.attrs.create(name="info", data=feature.info)
                dataset[index] = value

    def write_row_from_external(self, index: int, data_iter: typing.Iterable):
        """
        writes a single value into many paths given by the data_iter
        :param index: the location of the event
        :param data_iter: an iterable of tuples (hdf_path, value)
        """
        with h5py.File(self.context_data_file_path, "a") as file:
            for hdf_path, value in data_iter:
                dataset = file.get(hdf_path, None)
                if dataset is None:
```

```
68                      dataset = file.require_dataset(
69                          name=hdf_path, shape=(self.length,), dtype=type(value),
70                          chunks=True)
71                      # attention, no info string will be added
72                  dataset[index] = value
```

10.2.3 Feature Definitions

Listing 10: Implementation of event.py

```
"""This module contains the definition of the list of EvenDataFeatures
for the XBox2 data set."""
import typing
from functools import partial
import numpy as np
from src.utils.handler_tools.feature_class import EventDataFeature


def get_event_data_features(length) -> typing.Generator:
    """
    generates all custom EventDataFeatures for the XBox-2 data set
    :return: generator of features
    """
    pulse_length_info_text = "The pulse length is the pulse duration in mirco " + \
                             "seconds. The pulse is defined as the region where the " + \
                             "amplitude is higher than the threshold (= half of the " + \
                             "maximal value)"
    pulse_amplitude_info_text = "The pulse amplitude is the mean value of the " + \
                                "pulse. The pulse is defined as the region " + \
                                "where the amplitude is higher than the threshold " \
                                "(= half of the maximal value)"

    yield EventDataFeature(
        name="pulse_length", func=_pulse_length, length=length, output_dtype=float,
        hdf_path='PEI Amplitude', working_on_dataset='PEI Amplitude',
        info=pulse_length_info_text)
    yield EventDataFeature(
        name="pulse_amplitude", func=_pulse_amplitude,
        working_on_dataset='PEI Amplitude', length=length, output_dtype=float,
        hdf_path='PEI Amplitude', info=pulse_amplitude_info_text)

    yield EventDataFeature(
        name="pulse_length", func=_pulse_length, length=length, output_dtype=float,
        hdf_path='PKI Amplitude', working_on_dataset='PKI Amplitude',
        info=pulse_length_info_text)
    yield EventDataFeature(
        name="pulse_amplitude", func=_pulse_amplitude,
        working_on_dataset='PKI Amplitude', length=length, output_dtype=float,
        hdf_path='PKI Amplitude', info=pulse_amplitude_info_text)

    yield EventDataFeature(
        name="pulse_length", func=_pulse_length, length=length, output_dtype=float,
        hdf_path='PSI Amplitude', working_on_dataset='PSI Amplitude',
        info=pulse_length_info_text)
    yield EventDataFeature(
        name="pulse_amplitude", func=_pulse_amplitude,
        working_on_dataset='PSI Amplitude', length=length, output_dtype=float,
        hdf_path='PSI Amplitude', info=pulse_amplitude_info_text)

    yield EventDataFeature(
        name="pulse_length", func=_pulse_length, length=length, output_dtype=float,
        hdf_path='PSR Amplitude', working_on_dataset='PSR Amplitude',
        info=pulse_length_info_text)
    yield EventDataFeature(
        name="pulse_amplitude", func=_pulse_amplitude,
        working_on_dataset='PSR Amplitude', length=length, output_dtype=float,
        hdf_path='PSR Amplitude', info=pulse_amplitude_info_text)

    yield EventDataFeature(
        name="D1", func=_apply_func_creator(partial(np.quantile, q=.1)),
        working_on_dataset='DC Up', length=length, output_dtype=float, hdf_path='DC Up',
        info="calculates the first deciles of the data")
    yield EventDataFeature(
        name="D9", func=_apply_func_creator(partial(np.quantile, q=.9)),
        working_on_dataset='DC Up', length=length, output_dtype=float, hdf_path='DC Up',
        info="calculates the 9th deciles of the data")

    yield EventDataFeature(
        name="D1", func=_apply_func_creator(partial(np.quantile, q=.1)),
```

```
            working_on_dataset='DC Down', length=length, output_dtype=float,
            hdf_path='DC Down', info="calculates the first deciles of the data")
    yield EventDataFeature(
            name="D9", func=_apply_func_creator(partial(np.quantile, q=.9)),
            working_on_dataset='DC Down', length=length, output_dtype=float,
            hdf_path='DC Down', info="calculates the 9th deciles of the data")

    yield EventDataFeature(
            name="dc_up_threshold_reached", func=_dc_up_threshold_func,
            working_on_dataset="DC Up", length=length, output_dtype=bool, hdf_path="/",
            info="Decides if event is a breakdown with a threshold of -0.01 on the"
                 " DC Up signal. So if the min of DC Up is < (threshold assigned"
                 " by experts) it is labeled as a breakdown.")


def _get_data_above_half_max(data: np.ndarray):
    """
    returns data that has a value greater than half of its maximal value
    """
    threshold = data.max(initial=-np.inf) / 2
    return data[data > threshold]


def _dc_up_threshold_func(data: np.ndarray) -> bool:
    """
    checks if any of the signals is below a given threshold
    :param data: event data time series as an array
    """
    threshold = -0.05  # threshold defined by rf cavity experts
    return bool(np.any(data < threshold))


def _pulse_length(data: np.ndarray) -> float:
    """
    calculates the pulse duration in micro seconds of the region where the amplitude
    is greater than a threshold (= half of the maximal value)
    :param data: event data time series as an array
    """
    acquisition_window = 2.  # in micro seconds
    num_total_values = len(data)
    num_relatively_large_values = len(_get_data_above_half_max(data))
    if num_total_values == 0:
        pulse_length = 0.
    else:
        pulse_length = acquisition_window * \
                       (num_relatively_large_values / num_total_values)
    return pulse_length


def _pulse_amplitude(data: np.ndarray) -> float:
    """
    calculates the mean value where the amplitude is greater than a
    threshold (= half of the maximal value)
    :param data: event data time series as an array
    """
    return _get_data_above_half_max(data).mean()


def _apply_func_creator(func: typing.Callable) -> typing.Callable:
    """
    creates a feature-func that applies the input func to the event data array
    :param func: the func to apply
    :return: a function that applies func on event data arrays"""

    def apply_func(data: np.ndarray) -> float:
        """
        applies a func on event data arrays
        :param data: event data time series as an array
        """
        return func(data)

    return apply_func
```

Listing 11: Implementation of attribute.py

```python
"""This module contains the definition of the list of EvenAttributeFeatures
for the XBox2 data set"""
import typing
import numpy as np
import h5py
from src.utils.handler_tools.feature_class import EventAttributeFeature


def get_event_attribute_features(length: int) -> typing.Generator:
    """
    generates all EventAttributeFeatures for the XBox-2 data set
    :param length: number of values that will be calculated by each feature
    :return: generator of features
    """
    yield EventAttributeFeature(
        name="Timestamp", func=_get_timestamp, length=length, hdf_path="/",
        output_dtype=h5py.opaque_dtype('M8[us]'),
        info="The timestamp of the EventData is a property of the event group. It is "
             "given in a datetime format with the precision of micro seconds.")

    info_text = "These values originated from the Log_Type assigned by the" + \
                " CLIC-Team. Originally the Log_Type property had values in" + \
                " {0,1,2,3} where 0 stood for a healthy or normal log signal, and" + \
                " 3 for a breakdown. The label 1 and 2 stood for breakdown in 20ms" + \
                " and 40ms, which are the signals prior to a breakdown."

    yield EventAttributeFeature(
        name="is_healthy", func=fun_is_healthy_label, length=length,
        hdf_path="/clic_label/", output_dtype=bool, info=info_text)

    yield EventAttributeFeature(
        name="is_bd_in_40ms", func=fun_is_bd_in_40ms_label, length=length,
        hdf_path="/clic_label/", output_dtype=bool, info=info_text)

    yield EventAttributeFeature(
        name="is_bd_in_20ms", func=fun_is_bd_in_20ms_label, length=length,
        hdf_path="/clic_label/", output_dtype=bool, info=info_text)

    yield EventAttributeFeature(
        name="is_bd", func=fun_is_bd_label, length=length, hdf_path="/clic_label/",
        output_dtype=bool, info=info_text)

    yield EventAttributeFeature(
        name="log_type", func=fun_log_type, length=length, hdf_path="/clic_label/",
        output_dtype=int, info="These values originated from the Log_Type assigned "
                               "by the CLIC-Team. (is_healthy: 0, is_bd_in40ms: 1, "
                               "is_bd_in20ms: 2, is_bd: 3).")


def fun_log_type(attrs: h5py.AttributeManager) -> bool:
    """
    translates the 'Log Type' group properties (attributes) of the event data
    into a boolean value
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: value (is_healthy: 0, is_bd_in40ms: 1, is_bd_in20ms: 2, is_bd: 3)
    """
    label = attrs["Log Type"]
    return label


def fun_is_healthy_label(attrs: h5py.AttributeManager) -> bool:
    """
    translates the 'Log Type' group properties (attributes) of the event data
    into a boolean value
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: True if (is_healthy: 0, is_bd_in40ms: 1, is_bd_in20ms: 2, is_bd: 3)
        in other cases return False
    """
```

```
    label = attrs["Log Type"]
    if label in {0, 1, 2, 3}:
        is_defined_type = label == 0
    else:
        raise ValueError(f"'Log Type' label not valid no translation for {label}!")
    return is_defined_type


def fun_is_bd_in_40ms_label(attrs: h5py.AttributeManager) -> bool:
    """
    translates the 'Log Type' group properties (attributes) of the event data
    into a boolean value
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: True if (is_healthy: 0, is_bd_in40ms: 1, is_bd_in20ms: 2, is_bd: 3)
        in other cases return False
    """
    label = attrs["Log Type"]
    if label in {0, 1, 2, 3}:
        is_defined_type = label == 1
    else:
        raise ValueError(f"'Log Type' label not valid no translation for {label}!")
    return is_defined_type


def fun_is_bd_in_20ms_label(attrs: h5py.AttributeManager) -> bool:
    """
    translates the 'Log Type' group properties (attributes) of the event data
    into a boolean value
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: True if (is_healthy: 0, is_bd_in40ms: 1, is_bd_in20ms: 2, is_bd: 3)
        in other cases return False
    """
    label = attrs["Log Type"]
    if label in {0, 1, 2, 3}:
        is_defined_type = label == 2
    else:
        raise ValueError(f"'Log Type' label not valid no translation for {label}!")
    return is_defined_type


def fun_is_bd_label(attrs: h5py.AttributeManager) -> bool:
    """
    translates the 'Log Type' group properties (attributes) of the event data
    into a boolean value
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: True if (is_healthy: 0, is_bd_in40ms: 1, is_bd_in20ms: 2, is_bd: 3)
        in other cases return False
    """
    label = attrs["Log Type"]
    if label in {0, 1, 2, 3}:
        is_defined_type = label == 3
    else:
        raise ValueError(f"'Log Type' label not valid no translation for {label}!")
    return is_defined_type


def _get_timestamp(attrs: h5py.AttributeManager) -> np.ndarray:
    """
    returns the Timestamp from group properties/attribute in numpy datetime format
    :param attrs: the h5py.AttributeManager of an hdf.Group object
    :return: numpy datetime format of the timestamp
    """
    datetime_str = attrs["Timestamp"][:-1]
    return np.datetime64(datetime_str).astype(h5py.opaque_dtype('M8[us]'))
```

Listing 12: Implementation of trend.py

```python
"""This module contains the definition of the list of TrendDataFeatures
for the XBox-2 data set."""
from pathlib import Path
import typing
import numpy as np
import h5py
from src.utils.handler_tools.feature_class import TrendDataFeature


def get_trend_data_features(length: int, trend_data_file_path: Path) -> typing.Generator:
    """
    generates all TrendDataFeatures for the XBox-2 data set
    :param length: number of values that will be calculated by each feature
    :param trend_data_file_path: file path of the trend_data_file
    :return: generator of features
    """
    with h5py.File(trend_data_file_path, "r") as file:
        for key in file.keys():
            yield TrendDataFeature(
                name=key, func=_select(trend_data_file_path, key),
                output_dtype=h5py.opaque_dtype("M8[us]") if key == "Timestamp" else float,
                length=length, hdf_path="PrevTrendData",
                info=f"Previous trend data of {key}")


def _select(trend_data_file_path: Path, key: str):
    """
    creates and returns a function that selects trend data from given keys
    """
    def selection(indices_selection: np.ndarray):
        """
        returns a selection of one signal of the trend data
        :param indices_selection: selection of indices to return (array of boolean values)
        """
        with h5py.File(trend_data_file_path, "r") as file:
            return np.array(file[key])[indices_selection]
    return selection
```

Listing 13: Implementation of tsfresh.py

```python
"""This module contains the definition of the list of tsfresh features with predefined
settings for the XBox-2 data set."""
import pandas as pd
import tsfresh


def get_tsfresh(data: dict, settings: dict, n_jobs: int = 0) -> pd.DataFrame:
    """
    calls the tsfresh feature calculation on all input data
    combines all calculated features into a single dataframe and returns it
    :param data: a dictionary of {vec_name:data_vector}
    :param settings: the tsfresh feature settings
    :param n_jobs: the number of cores used to calculate the features
        using multiple cores when calculating a few features may cause a big overhead
    :return: dataframe of calculated features
        columns are the input data vector names
        indices are the feature names
    """
    df_molten = pd.DataFrame()
    for length in {val.shape[0] for val in data.values()}:
        df_tmp = pd.DataFrame({key: val for key, val in data.items()
                               if len(val) == length})
        df_tmp['column_sort'] = df_tmp.index
        df_molten = df_molten.append(
            df_tmp.melt(id_vars='column_sort', value_name="tsfresh", var_name="channel"))

    df_ret = tsfresh.extract_features(timeseries_container=df_molten,
                                      column_id="channel",
                                      column_sort="column_sort",
                                      column_value="tsfresh",
                                      default_fc_parameters=settings,
                                      n_jobs=n_jobs,
                                      disable_progressbar=True).T
    return df_ret
```