



UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING

MASTER THESIS

Sensor Fusion using Signal Processing and AI Tools Applied to CERN-CMS Detector

Hasan Aljamea

Supervised by
Dr. Zouhir Bahri
Dr. Mehmet Özgür Sahin

A thesis submitted in partial fulfillment of the requirements
for the degree of **M.Sc. in Artificial Intelligence Systems**

May 19, 2024

Defense Committee's Approval

University of Bahrain
College of Engineering
Department of Electrical and Electronics Engineering
Master of Science in Artificial Intelligent Systems

Student Hasan Haider Aljamea ID number 202012499 has defended his thesis titled " Sensor Fusion using Signal Processing and AI Tools Applied to CERN-CMS Detector" by which he has fulfilled the requirement for the degree of Master of Science in Artificial Intelligent Systems on Wednesday, May 15, 2024.

The committee recommends awarding of M.Sc. Degree with the grade of: Pass.

(Defense Committee)

Name

Signature

Asst. Professor Zouhir Bahri/Ozgur Sahin Supervisor

Ozgur Sahin

Professor Waleed Fakhr

External examiner

Mohamed Waleed Fakhr

Professor Alauddin AlOmari

Internal examiner

AlaOmari

Academic department's stamp





Dissertation Deposit Form

Author Name:	Hasan Haider Aljamea (202012499)		
Email:	202012499@stu.uob.edu.bh	Contact No:	32206122
College:	Engineering		
Department:	Electrical & Electronics Engineering		
Thesis Title:	Sensor Fusion using Signal Processing and AI Tools Applied to CERN-CMS Detector		
Year:	May, 15 2024		

Author's Declaration: I agree to the following conditions:

The above thesis will be available (in the University of Bahrain library and externally) and reproduced as necessary at the discretion of the University of Bahrain. It may also be digitized by the University of Bahrain and made available via the university's repository or via other parties on the Internet.

This condition shall apply to ALL copies including electronic copies.

The above thesis has been provided on the understanding that it is copyright protected material and that no quotation from this thesis may be published without proper acknowledgement.

Signature of the Author:		Date:	20/5/24
--------------------------	---	-------	---------

Abstract

This thesis explores the application of Artificial Intelligence (AI) and Signal Processing (SP) techniques in the context of Sensor Fusion applied to the high-energy physics experiment at the Large Hadron Collider (LHC) of the European Centre for Nuclear Research (CERN). The central aim is to help in maintaining the integrity of the massive sensor array used at the Minimum Ionizing Particle Timing Detector (MTD) located at the inner layers of the Compact Muon Solenoid (CMS) and develop a user friendly platform to collect and compactly present the large sensor data to the monitoring operator. Hence, a major part of this work aims at detecting and raising warnings of potential operational issues before they escalate by applying different AI techniques for anomaly detection, namely various autoencoder architectures including Convolutional Neural Network (CNN), 2D Convolutional with Long Short-Term Memory (Conv2DLSTM), CNN with U-net architecture, and Conv2DLSTM with U-net architecture. The latter method turned out to provide the best performance with an accuracy of 96.2% on simulated data. Computer simulations indicate that this algorithm is quite robust against additive noise as its performance was found to degrade by only about 1% with a Signal-to-Noise Ratio (SNR) decreasing from 17dB to 5dB. The wavelet denoising algorithm was investigated to possibly mitigate this noise effect in light of the harsh operating environment at CMS with a marginal success since for low SNR sensor signals it caused the anomalies to be hidden hence missed by the algorithm. Obtaining practical data from the sensor array was a rather challenging task, and as of the writing of this work only partial data was obtained. Initial results of applying the algorithm to the partial real data in hand showed a comparable accuracy of 94.5 %. The MTD services developed in this work using FastAPI Python library and xDAQ C++ Framework as well as the proposed anomaly detection methods should contribute to safely running the MTD close to its optimum point.

Keywords — CERN, CMS, MTD, DAQ, Anomaly Detection, Incremental Learning

Acknowledgments

I extend my heartfelt gratitude to the following individuals and institutions whose support and guidance have been invaluable in the completion of this thesis:

Dr. Zouhir Bahri, for his unwavering academic support and for recognizing the potential in me, which led to his recommendation for the joint UOB-CERN-CMS project. His supervision and mentorship have been instrumental in shaping my academic journey.

Dr. Mehmet Özgür Sahin, whose guidance and encouragement were instrumental in my role as a productive member of the MTD DAQ team. His insights and directives significantly contributed to the progress and success of this project.

The University of Bahrain and Sh. Ebrahim bin Mohammed Al Khalifa Center for Culture and Research for providing the funding that enabled me to stay at CERN for two years needed to complete the project. Their support played a crucial role in the realization of this endeavor. I am deeply thankful for their support, encouragement, and belief in my abilities throughout this academic endeavor.

Contents

Abstract	iii
Acknowledgments	iv
List of abbreviations	viii
List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 Motivation and Objectives	1
1.2 Literature review	1
1.3 Work Contributions	3
2 Background	4
2.1 The CMS detector	4
2.2 CMS Phase II Upgrade	4
2.3 MIP Timing Detector (MTD)	5
2.3.1 Data Acquisition systems and monitoring systems	5
2.3.2 MTD DAQ Electronics	6
2.4 MTD DAQ Software	8
2.4.1 ASIC chip classes abstraction	8
2.4.2 MTD DAQ Services	9
2.4.2.1 MTD Configurator	11
2.4.2.2 MTD Controller	11
2.4.2.3 MTD Collector	11
2.4.2.4 MTD GUI	11
3 MTD DAQ Anomaly Detector	14
3.1 The anomaly definition	14
3.2 Anomaly Detection Algorithms	15
3.2.1 Anomaly Detection methods selection criteria	15
3.2.2 Pre-processing: Re-scaling and Denoising	16
3.2.2.1 Re-scaling	16
3.2.2.2 Denoising	16

3.2.3 Auto-encoder models	17
3.2.3.1 ConvReshape Layer	17
3.2.3.2 AutoEncoders architectures	17
3.2.4 One-Class classification methods	19
3.2.4.1 AutoEncoder's MSE	19
3.2.4.2 k-means one-classification	19
3.3 Anomaly Detection Simulation	20
3.3.1 Dataset	20
3.3.2 Activity simulation	21
3.3.3 Anomaly simulation	22
3.4 Real Data	23
4 Results and recommendations	24
4.1 Results On Simulated Data	24
4.2 Results On Real Data	26
5 Conclusion	28
5.1 Summary	28
5.2 Future points of expansion	28
Bibliography	30
Appendix A: Formal Mathematical Derivations	33
A.1 Discrete form of the Heat equation	33
A.2 Updating variance σ^2 and mean μ of scalar data	35
A.3 Updating Covariance matrix and mean of vector data	36
Appendix B: BTL Readout Unit full sensor list	39
Building the AutoEncoders using Tensorflow	41
Appendix B: BTL Readout Unit full sensor list	41

List of Abbreviations

ALDO2	TOFHIR2 chip Voltage regulator
ATCA	A dvanced T elecommunications C omputing A rchitecture
BTL	B arrel T iming L ayer
CC	C onnector C ard
CDR	C entral D ata R ecording
CMS	C ompact M uon S olenoid
D2S	D ata t o S urface
DAQ	D ata A cquisition
DCS	D etector C ontrol S ystem
DTC	DAQ and T rigger C oncentrator
DTH	DAQ and T iming H ubs
DCR Noise	D ark C urrent N oise due to R adiation damage
ETL	E nd cap T iming L ayer
EVM	E vent M anager
FEB	F ront E nd B oard (context: HCL electronics)
FPGA	F ield- P rogrammable G ate A rray (context: Integrated Circuits)
EOS	Name of the IT solution for large amounts of physics data and user files and other services
HCAL	H adron C alorimeter L ayer
HIF	H CAL I ntegration F acility
HLT	H igh L evel T rigger processors
HPC	H igh P erformance C omputing clusters
HPD	H ybrid P hotodiodes
lpGBT	low-power Giga-Bit transceivers
LHC	L arge H adron C ollider

LHCC	L arge H adron C ollider C ommittee
Luminosity	Measure of how many collisions are happening in a particle accelerator
LYSO crystal	LYSO (Lu 1.8Y. 2. SiO ₅ :Ce) is a Cerium benefits compared to many common scintillation doped Lutetium-based scintillation crystal that offers several materials. LYSO offers high density and short decay time.
MIP	M inimum I onizing P article
MTD	MIP T iming D etector
SCR	S urface C ontrol R ooms
SiPM	Silicon photomultipliers, often called "SiPM" in the literature, are solid-state single-photon-sensitive devices (context: used in detector chips)
TOFHIR	T ime- O f- F light H igh R ate
XMAS	X DAQ M onitoring and A larm S ystems
Pseudo-rapidity (η)	In experimental particle physics, is a commonly used spatial coordinate describing the angle of a particle relative to the beam axis.
USC	U nderground C ontrol room
UXC	U nderground C avern room
xDAQ	D ata A cquisition software
XaaS	A general, collective term that refers to the delivery of anything as a service.

List of Figures

2.1 CMS Cross-section highlighting MTD location [32]	5
2.2 High pile-up event with 78 reconstructed vertices taken in 2012 by CMS [34]	5
2.3 Conceptual design of the overall CMS DAQ [33]	6
2.4 BTL Compoments [35]	7
2.5 MTD Barrel frame where trays will be fitted	7
2.6 BTL Readout Unit (aka Concentrator card)	8
2.7 locations of sensors whose readings are collected by CC	9
2.8 Python Chip ASIC chip classes abstraction [37]	9
2.9 MTD Services. [38]	10
2.10 MTD GUI Command page	12
3.1 Initial Frame ($t_n = 0$) sensor window simulations with and without Anomaly	15
3.2 AutoEnocders	18
3.3 U-net AutoEnocders	18
3.4 MSE Autoencoder One-class Classification Algorithm	19
3.5 K-means One-class Classification Algorithm	20
3.6 BTL Readout Units representation as a sensor window of $(C \times L)$	21
3.7 One-minute excerpt of sensor reading	23
4.1 Anomaly detection results using K-means one-class classification	25
4.2 Anomaly detection results using MSE one-class classification	25
4.3 Noise Robustness test results for LSTMUnet-MSE	25
4.4 Noise Robustness test results for LSTMUnet-MSE with denoise preprocessing	26
4.5 Anomaly detection results using k-means one-class classification on real data	26
4.6 Anomaly detection results using MSE one-class classification on real data	27

List of Tables

2.1 Summary of the BTL modularity and channel count. 33	8
2.2 BTL Readout Unit Sensor Counts 33	8
3.1 Wavelet parameters selection ranges (Gene Space) and solution	17
3.2 Wavelet Functions from <code>pywt.wavelist(kind='discrete')</code>	17
3.3 SNR levels used in the robustness tests	21
3.4 Sesnors that were included in real data	23
B.1 BTL Readout Unit full sensor list	39

Introduction

1.1 Motivation and Objectives

The Compact Muon Solenoid (CMS) [1] experiment is one of the megaprojects that are being carried out at the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN) [2]. The CERN project is an international consortium comprising over 40 member countries and hundreds of affiliated universities, as well as thousands of scientists and researchers from around the world, to investigate various phenomena related to particle physics. Particle physicists at CMS study proton-to-proton collisions to understand how fundamental particles interact. To that end, a large amount of collision data is collected for analysis. This data improves scientists' understanding of the universe's fundamental particles and interactions.

This relentless search for new discoveries demands an ever-increasing rate of collisions to increase the likelihood of capturing rare events. To achieve this goal, the LHC is to be upgraded to a high-luminosity Large Hadron Collider (HL-LHC), where the Levelled luminosity can be up to $5.0 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$ [3]. The construction is scheduled to be implemented during the Long Shutdown III (LS3) LHC maintenance break estimated to start in 2026 to 2029 [4], a significant milestone in the project's timeline when crucial maintenance activities occur. The rising collision rate makes tracing and isolating collision effects harder, greatly hindering physics analysis performance. To address the challenges posed by the escalating collision rate, a new layer known as the Minimum Ionizing Particle Timing Detector (MTD) is incorporated into the CMS system. This layer provides a very precise time measurement of the minimum ionizing particle's (MIP) passage, which is between 30 and 60 picoseconds (ps) [5]. MIPs such as muons, electrons, and pions emit secondary photons when they pass through crystals like silicon photomultipliers (SiPM) and Low-Gain Avalanche Diodes (LGAD) [6, 7]. The precise timing measurements enhance the collision recording by introducing a temporal dimension to the data, enabling the tracking and isolation of collisions.

The MTD detector includes electronics positioned on the cylindrical barrel known as the barrel timing layer (BTL) and the end-cap timing layer (ETL) at the detector's ends. The MTD electronic system comprises approximately 66 thousand diagnostic sensors, including temperature, voltage, and current. These sensors help monitor the functionality of MTD data acquisition (MTD DAQ). The large number of sensors presents a major challenge for human operators who must analyze and interpret extensive sensor data. Hence, an important initial task of this work consists of developing the MTD services software that offers a centralized system for fusing and compactly presenting the massive sensor data towards controlling and monitoring the MTD operations, keeping it safely operating close to its optimum point. Another important goal of this work is to maintain the integrity of the large sensor array by detecting anomalies and triggering corrective measures before possible problems escalations occur. This is particularly important in light of the very harsh operation environment the sensors are subjected to, namely extreme temperature and radiation. It's important to note that this project does not address the resolution or correction of anomalous behavior but rather points out its occurrence.

1.2 Literature review

The MTD/CMS may be considered one of the world's most complex research facility systems. Exceeding 66,000 in number, the MTD's gigantic array of sensors constitutes in itself several challenges. First, monitoring all the sensors simultaneously is difficult for humans due to the huge amount of data. Hence, there is a need to efficiently and compactly gather and represent the massively generated data in a clear and user-friendly way for the operating monitors. Such Sensor Fusion [8] not only provides efficiency and compactness in data handling but also improves

reliability [9] by combining data from multiple sensors instead of relying on individual inputs that are prone to malfunctions. Secondly, and in turn, there is a need to maintain the integrity of these sensors by being able to detect possible anomalies in their signals.

In a broad sense, the topic of anomaly detection falls under the general problem of a one-class classification. Several methods have been used, such as the k-means [10], one-class Support Vector Machine (SVM) [11], Isolation forest (IF) [12], Local Outlier Factor (LOF) [13], and the Autoencoders [14, 15, 16, 17, 18, 19].

Autoencoders are neural networks that extract features from the sensor data and fuse it in a lower-dimensional latent space that ideally should remove any noise while preserving essential information needed to detect abnormal data acquisition operations. As such, autoencoders may be viewed as another layer of sensor fusion from the integrity assurance point of view [20, 21, 22]. Owing to the harsh operating environment at the CMS experimental cavern, namely extreme temperature and radiation stress, it is important to consider a robust classification method for noise and disturbances. Since the autoencoders inherently possess denoising capabilities [23, 24], they have been selected in this work. Autoencoders come with different architectures, such as Convolutional Neural Network (CNN) [25], 2D Convolutional with Long Short-Term Memory (Conv2DLSTM) [26], CNN with U-net architecture [27], and Conv2DLSTM with U-net architecture [28].

While earlier reports on Anomaly detection at CMS subsystems [14, 15, 16, 17, 18, 19] have been published, they have not been specifically applied to data acquisition. Furthermore, these approaches were rather complex in the context of their applications. Anomaly detection techniques in the CMS experiment span various sections and subsystems with many applications. Some of the examples illustrating the wide range of applications related to anomaly detection at the CMS experiment include Silicon sensors inspection for defects [14], Managing experimental data quality [18, 19], Monitoring the health of the Hadron Calorimeter detector, and aiding the trigger system in identifying new events [15, 16].

The silicon sensors were visually inspected by overlaying the silicon images onto the model, which allowed for identifying their specific placement within the image. Due to the large quantity of those sensors, it would have been an arduous undertaking for a human. [14]. Monitoring the health of the Hadron Calorimeter detector is context-dependent and requires forecasting. A Time Series Forecasting (TSF) cascaded from sequence to sequence (S2S) autoencoder was employed to handle the additional needs, given the increased complexity of the model. The model is trained using sensor readings from normal operations and the operational environment. The model's forecasting capability facilitates proactive maintenance, allowing for early detection and resolution of issues before their occurrence [17]. Reconstructing the particles involved in the collisions seen by the sensors requires analysis of the CMS experiment data. Validating the results is required to guarantee correctness and reliability. Human data quality inspector specialists analyze data quality using histograms to make decisions. If there is an abnormality, histograms should display a significant divergence from the expected shape. Alternately, VAE autoencoders detect anomalies in the data [18, 19].

The trigger system consists of a level-1 trigger system that is subsystem-specific, usually hardwired electronics since it needs to respond rapidly (around 5 ms), and the global trigger system takes readings from all subsystems hosted in high-computing clusters and makes global decisions for the entire system. Whether or not to record the event is the trigger's choice. Since it is not feasible to store all the events, the trigger system must carefully select which events to record based on specific criteria or conditions. The conditions are based on the event's novelty since common events studied are not worth the storage space to record them. One example of anomaly detection application in the global trigger system is discovering novel events that the trigger system can allow to be recorded [15, 16].

The scope of this work falls under the main category of monitoring the health of the detector, but in a simpler approach and specifically applied for the first time to the MTD DAQ subsystem. Hence, the primary goal of the MTD DAQ anomaly detector is to trigger an alarm in case it deviates from the normal operating point. As a starting phase and till the time of writing of this thesis, the proposed anomaly detection only deals with simple binary status (normal/abnormal operation). However, as the MTD DAQ evolves, new requirements may emerge, resulting in more complex anomaly detector models. This complexity is necessary in order not only to detect the occurrence of anomalies but also to identify the corresponding malfunctioning sensors and incorporate context into the anomaly detection process. Although the requirements and implementation of the proposed model are relatively simple compared to others, it is still presented with some challenges such as the dataset size, where each data point includes time frames from all BTL readout units, also known as Concentrator cards (CC). This necessitates training the model in *batches* since it is unfeasible to load and handle the entire dataset at once. This is one of the main reasons that tilted the balance towards using autoencoders in the anomaly detection model in addition to their inherent denoising capabilities.

Finally and in the context of mitigating the harsh operating environment of the sensor network at the MTD, this work explores the possible merits of pre-processing the sensor data before feeding it to the anomaly detection system.

Hence, this work considers using the outstanding Wavelet-based denoising method [29] in order to alleviate the problem of noise corrupting the sensor data.

1.3 Work Contributions

The main contributions of this work include:

- An extensive set of MTD services were originally developed using Python and C++ to perform:
 - MTD configurator.
 - MTD Controller.
 - MTD Collector.
 - Sensor Data and MTD Graphical User Interface.
- Generate an extensive set of simulated sensor data to mimic realistic working conditions for the various types of sensors.
- Assist in gathering partial real MTD data.
- Implemented various autoencoder architectures, namely CNN, Conv2DLSTM, CNN-U-net, and LSTM-U-net.
- Implemented a convolutional layer to accommodate different data shapes.
- Implemented multi-batched K-means in the context of single-class classification via derived updated scalar and vector statistics.
- Carried out extensive comparative performance analysis of the different autoencoder architectures to point out the method of choice, namely LSTM-U-net.
- Performed a robustness analysis of the selected model in light of additive noise.
- Investigated the merit of using the wavelet denoising method further to alleviate the effect of noise in the sensor data.
- Tentatively tested the proposed model on partial real MTD data.
- Contributed in developing the Slow-Control Adapter (SCA) Chip python wrapper API class.
- The derivation of the statistics updating algorithm, namely the scalar and vector data mean and covariance that was used for the efficient implementation of multi-batching in K-means.
- Contributed to the MTD DAQ hardware setup enabling the readout of the sensor signals from the front-end ASICs. This process involves mounting the BTL read out units into the tray, connecting the BTL modules through the cooling plate, and connecting it to the serenity board with proper connectivity verification.

The remaining of this thesis is organized as follows. Chapter 2 provides some background material related to the MTD hardware and electronic systems, pointing out the sensor array placement. This chapter also summarizes the initial efforts of this project that have contributed to the MTD DAQ hardware setup enabling the readout of the sensor signals in addition to developing various services needed to gather and fuse the massive sensor data in a compact and user-friendly format for the monitoring operator. Chapter 3 details the work related to ensuring the integrity of the sensor array by applying different AI techniques for anomaly detection. This chapter also examines the robustness of the proposed algorithm against noise and discusses possible mitigation approaches. Chapter 4 provides the results and recommendations of this work. Finally, Chapter 5 summarizes this work and provides future points of expansion.

Background

Since the initial aim of this work is to help in the data gathering and fusion, it is important to provide an overview of the hardware platform of the MTD. This chapter is dedicated to summarize the various stages and electronic systems of the MTD, in particular the sensor array that is the focus of this work.

Being familiar with the MTD DAQ electronics should provide a clearer context for the diagnosis of the sensors used in the DAQ anomaly detector. Hence, the services developed in the context of gathering and monitoring the sensor data include:

- The MTD Configurator: with the role of editing the MTD electronics configurations
- The MTD Controller: implements the configuration and control commands
- The MTD Collector: with the aim of collecting the monitoring data. Moreover, this service enables the seamless integration of MTD DAQ into the MTD GUI portal, offering a user-friendly interface that enhances the monitoring and management of the MTD system.

The above services allow for the efficient monitoring and management of the MTD system, ensuring optimal functionality and performance. This in turn, should Establish the foundation for the DAQ anomaly detector to operate effectively, and offer early warnings of deviations from optimal data acquisition conditions, hence empowering users to take necessary actions within the same MTD GUI webpage portal.

2.1 The CMS detector

The world's largest particle accelerator was constructed at CERN, where operations started on September 10, 2008^[30], to study fundamental particles. Proton beams in the Large Hadron Collider (LHC) collide at specific points around the accelerator ring where experiments like ATLAS, CMS, Alice, and LHCb are conducted. The collisions of proton bunches are made possible by a 27-kilometer ring of superconducting magnets with accelerating structures ^[31].

The Compact Muon Solenoid (CMS) experiment weighs 1400 tons, making it the heaviest in the LHC. The CMS experiment consists of tightly packed sub-detectors, with a high-magnet solenoid separating the muon detectors from the other components. The CMS experiment is cylindrical, measuring 28.7m in length and 15m in diameter, with multiple subdetectors arranged in an onion-like structure (refer to figure ^{2.1}).

2.2 CMS Phase II Upgrade

Particle physicists constantly seek to complete the standard model by discovering more about the universe's fundamental building blocks. Particle physicists and researchers are working hard to identify and observe rare interactions that are yet to be detected. In that pursuit, the luminosity of particle accelerators has been increased to boost the number of collisions, consequently increasing the likelihood of novel interactions occurring. With those novel interactions, particle physicists may have more insight into the fundamental blocks of the universe. Achieving this luminosity requires a significant detector upgrade in all experiments, CMS included. In addition to hardware radiation and low-temperature tolerance, new modules have been added to maintain the same physics performance, such as the Minimum Ionizing Particle Timing Detector (MTD). The technical report "The Phase-2 Upgrade of the CMS Data

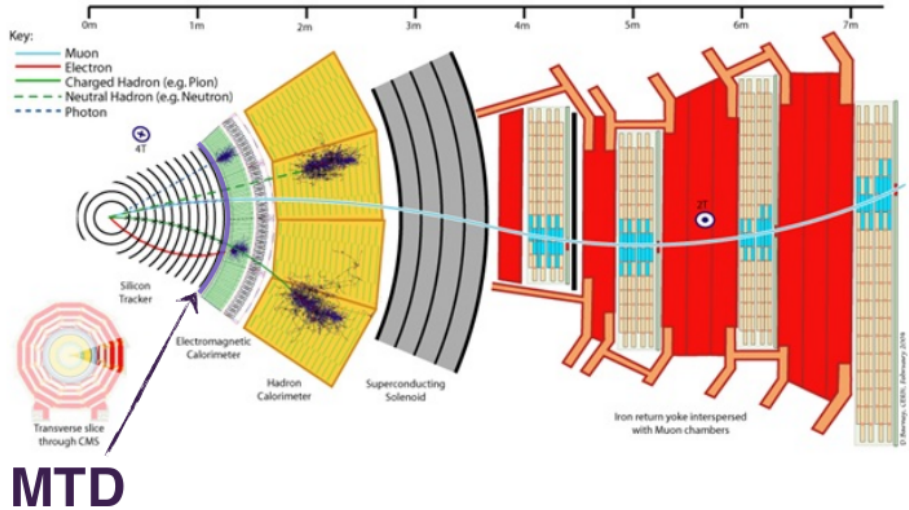


Figure 2.1: CMS Cross-section highlighting MTD location [32]

"Acquisition and High-Level Trigger" outlines the CMS upgrade plan" [33]. This upgrade will lead to new heights in understanding of fundamental physics to be integrated into the CMS during the Long shutdown (LS3) in preparation to be used during RUN 4.

2.3 MIP Timing Detector (MTD)

The Minimum Ionizing Particle (MIP) Timing Detector (MTD) is a new addition to the CMS experiment introduced in the Phase II upgrade. As the LHC luminosity increases, collisions will also increase to 40 MHz collision frequency. Consequently, isolating a collision event from its accompanying interactions, known as pileup (Figure 2.2), will become more challenging. Identifying and isolating collisions from the rest of the pile-up is crucial for observing the physics of subatomic particles. Hence, the inability to do so directly impacts physics performance. Therefore, the MTD Timing layer is introduced to improve the physics performance by providing precise timing measurements of charged particles (MIP). With the timing data, collision events can be isolated within the MTD timing resolution, which is currently between 50 and 60 picoseconds.

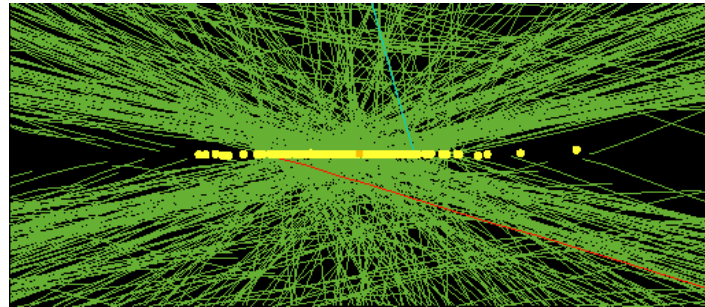


Figure 2.2: High pile-up event with 78 reconstructed vertices taken in 2012 by CMS [34]

2.3.1 Data Acquisition systems and monitoring systems

This section will focus on the Data Acquisition process employed in MTD BTL (Barrel Timing Layer), a project in which I have participated and made contributions to. This will provide an overview of the MTD DAQ electronics, beginning with the on-detector sensors and concluding with the data transmission and collection.

Figure 2.3 depicts the Conceptual design of the overall CMS DAQ. A data acquisition system (DAQ) is primarily responsible for performing several crucial functions. First and foremost, its primary function is to collect data from

the detector sensors located in its surroundings. After the data are collected, they are sent to the Level-1 Trigger control and the surface. The Level-1 trigger control mechanisms initiate specific actions or responses based on the sensor reading obtained by the DAQ. Furthermore, the data will be transmitted to the surface and stored in a designated storage system for future reference and analysis.

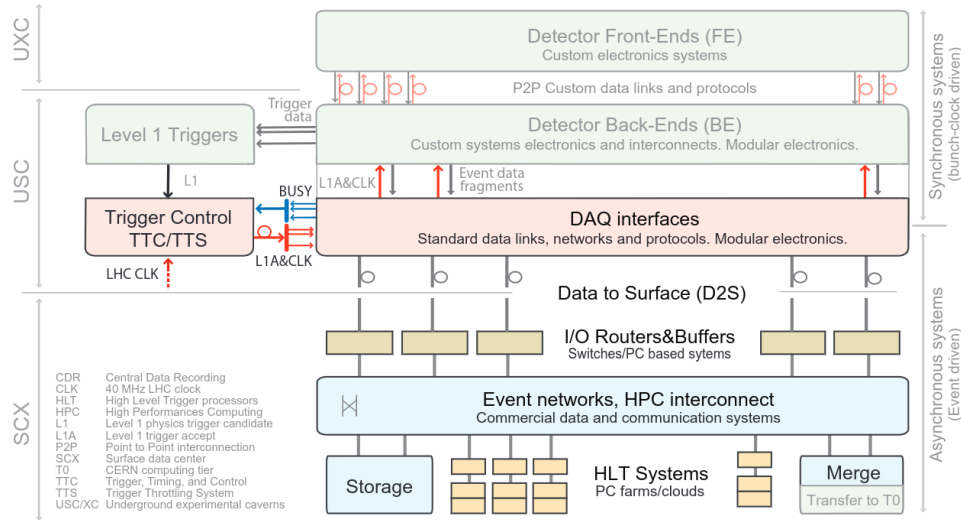


Figure 2.3: Conceptual design of the overall CMS DAQ [33]

In addition, a data acquisition system is responsible for ensuring the seamless operation of the data acquisition process, which involves monitoring the sensors and the data transmission process to guarantee optimal operation. Furthermore, the system is equipped with control mechanisms to make necessary adjustments or interventions in case any problematic abnormalities are detected during the data acquisition process without physically entering the CMS experiment cavern. After the CMS assembly, the experimental cavern will remain closed until the subsequent scheduled long-term maintenance shut-down. Thus, the controls and adjustments are performed remotely while the beam is active, utilizing the pre-assembled system.

2.3.2 MTD DAQ Electronics

The Barrel Timing Layer (BTL) is a cylindrical framework that contains the BTL components arranged in a grid pattern in the inner side of cylinder. Let us define the cylindrical grid as $(n\phi \times mZ)$, where n represents the number of components along the circumference and m represents the number of components along the barrel length.

The BTL components are positioned in trays that are organized in a $(36\phi \times 2Z)$ configuration (see figures 2.4, 2.5). Each tray consists of six Readout units arranged in a $(1\phi \times 6Z)$ configuration. The Readout unit consists of a Concentrator card, a cooling plate, and 12 BTL frontend modules arranged in a $(3\phi \times 8Z)$ configuration. Each BTL Frontend module is equipped with a TOFHIR ASIC chip. The TOFHIR ASIC chip comprises 16 crystals, resulting in 32 channels, see tables 2.1, 2.2, B.1. The SiPm channels are responsible for detecting the MIP signal in the crystal. Therefore, each crystal possesses two channels for calculating the time it takes for an object to travel.

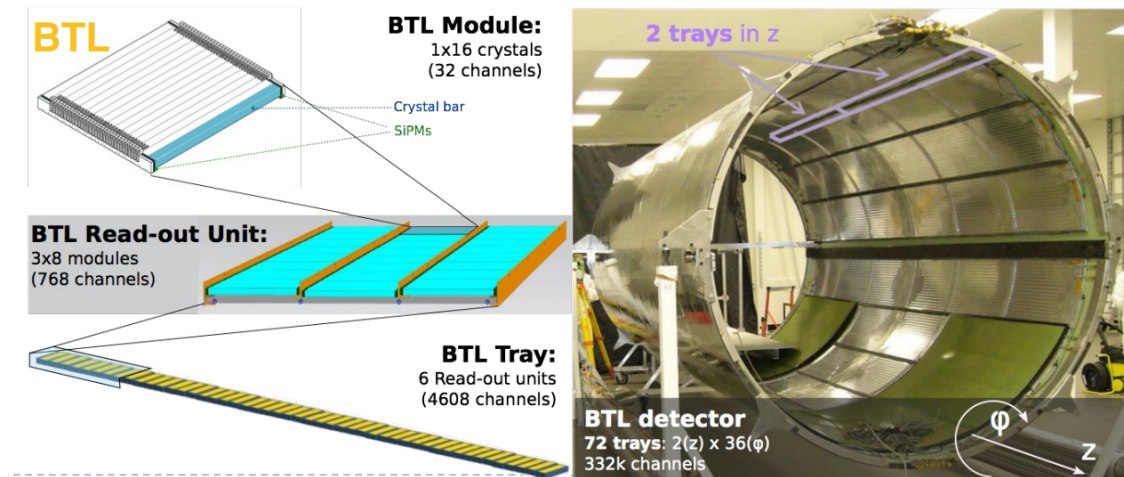


Figure 2.4: BTL Components [35]



Figure 2.5: MTD Barrel frame where trays will be fitted

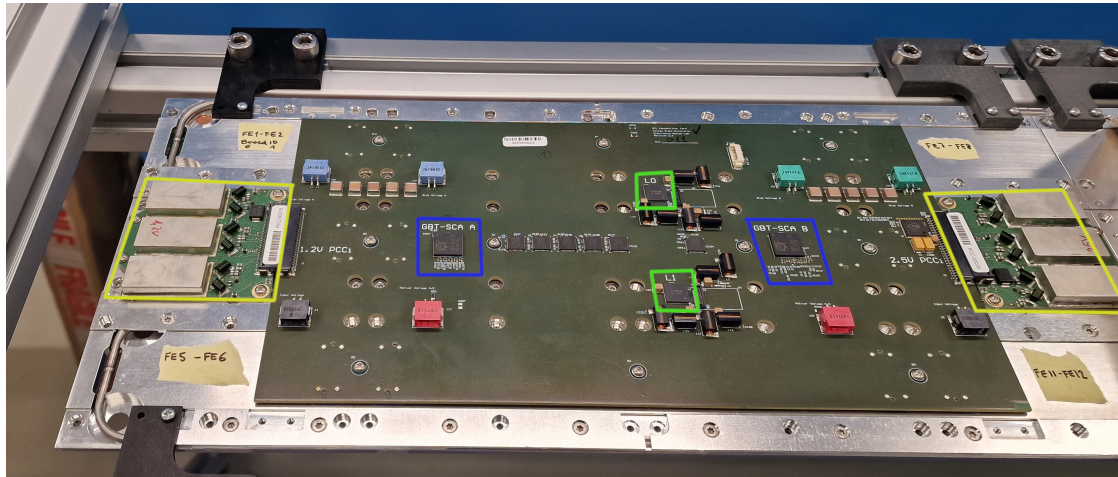
Table 2.1: Summary of the BTL modularity and channel count. [33]

	Module	RU	Tray	Total
Channels (SiPMs)	32	768	4,608	331,776
Crystals	16	384	2,304	165,888
ASICs	1	24	144	10,368
Modules	-	$3 \times 8 = 24$	$3 \times 48 = 144$	$108 \times 96 = 10,368$
Readout units (RU)	-	-	$1 \times 6 = 6$	$36 \times 12 = 432$
Trays	-	-	-	$36 \times 2 = 72$

Table 2.2: BTL Readout Unit Sensor Counts [33]

Source	Description	Nominal Range	Count
On CC board	CC board Temperature	Pt1000	3
PCC	Power Card Converter	Pt1000	4
Front-end Modules	SiPM	Pt1000	$8 \times 2 = 16$
Front-end Modules	Current Bias	(0-2.247) mA	$12 \times 4 = 48$
SCA	Chip Voltage	(0-2.5)v	2
LPGBT	Chip Voltage	(0-2.5)v	5
LPGBT	Vin Voltage	(0-15)v	1
			78

The concentrator cards, also known as BTL Read-out Units, house the MTD electronics responsible for reading and transmitting sensor data to the MTD back-end systems situated in the underground cavern chamber (UXC). The temperature, voltage, and current sensors are present in various components of the low-power GigaBit Transceiver (lpGBT), Slow-Control Adapter (SCA), Power Card converters (PCC), and the concentrator card itself. Figure 2.7 illustrates the positions of these sensors. Additionally, figure 2.6 provides a visual representation of the sensor locations specifically on the concentrator card.



Blue : Slow control adapter (SCA)
 Green : low power Gigabit transceiver (lpGBT)
 Yellow: Power Converter Card (PCC)

Figure 2.6: BTL Readout Unit (aka Concentrator card)

2.4 MTD DAQ Software

2.4.1 ASIC chip classes abstraction

The MTD DAQ team has created a custom software to carry out data acquisition and maintenance using MTD DAQ ASICs. These ASICs are programmed using VHDL FPPGA programming, C++, and are wrapped in a Python

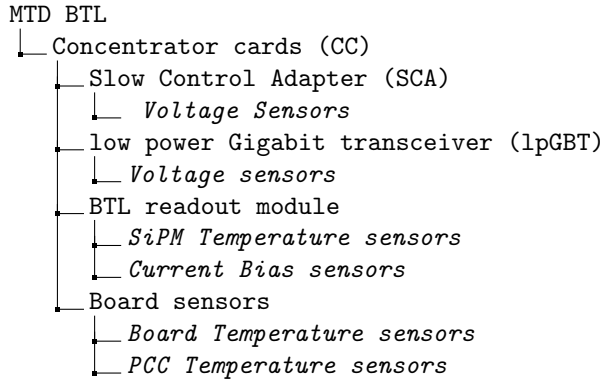


Figure 2.7: locations of sensors whose readings are collected by CC

interface that activates the lower-level functions.

Utilizing Python’s readability and user-friendliness, a well-structured hierarchical structure was designed for ASICs operations. This approach avoids the drawback of Python’s slow run-time by executing the operations through calling pre-compiled codes.

Expanding upon the EMP FPGA serenity framework [36], the low power Gigabit transceiver (lpGBT), Slow Control adaptor (SCA), and Vortex plus (VTRx+) operations of the ASICs chips have been streamlined by inheriting all shared operations from the chip class. The location of the ASICs on the BTL readout unit, also referred to as the Concentrator card (CC), with their Python class abstraction are illustrated in figure 2.8

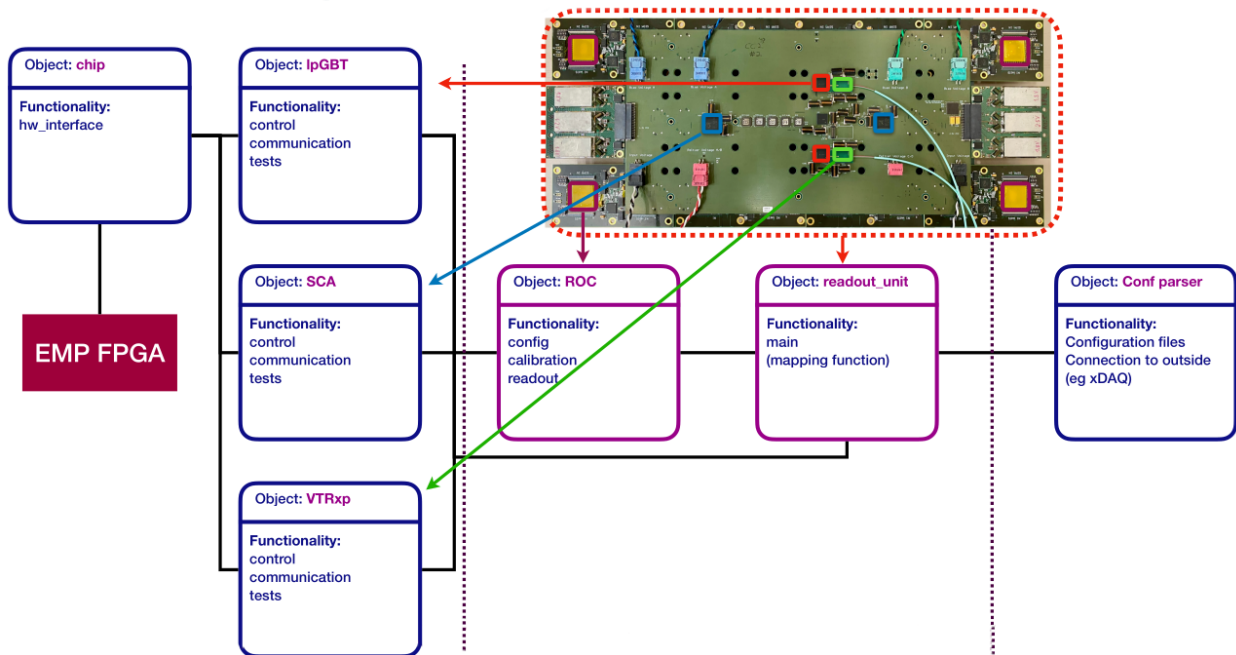


Figure 2.8: Python Chip ASIC chip classes abstraction [37]

2.4.2 MTD DAQ Services

MTD DAQ Services have been the main focus of my contribution. Where they play a pivotal role in managing and operating MTD DAQ systems, to which I have made some significant developments. These systems serve as the fundamental support for different sectors of MTD DAQ electronics. My contribution involved designing and implementing a centralized framework for managing MTD electronics. Furthermore, the service regularly provides updates and enhancements to optimize the performance and functionality of the data acquisition processes and system

monitoring. The enhancements strengthen the durability and effectiveness of the MTD DAQ system, highlighting my technical and strategic input to the project. Figure 2.9 provides an overview of the MTD Services systems and communications, which will be examined in depth.

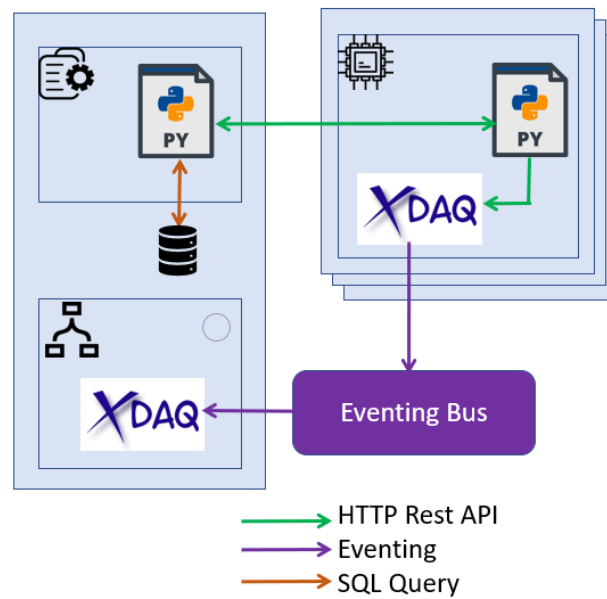


Figure 2.9: MTD Services. [38]

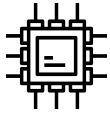
2.4.2.1 MTD Configurator



Centralized Control is attained by managing all MTD electronics from a single, convenient location by utilizing the MTD Configurator. In order to achieve this objective, the FastAPI framework [39] is used to create the Python interface responsible for transmitting and handling HTTP/Restful requests between MTD controllers. Furthermore, it is linked to the configuration database, which stores all MTD configurations. SQLAlchemy was used to establish, modify, and retrieve data from the configuration database, offering versatility in selecting the database and facilitating the transition from configuration files to populating the configuration database. The MTD DAQ GUI enables the visualization of MTD configurations and facilitates the transmission of commands to MTD controllers. The user-friendly centralization of management is accomplished by employing the MTD GUI (refer to section 2.4.2.4) as the graphical user interface, while web back-end development is conducted in the MTD configurator. After successful testing, the back-end development facilitates the seamless integration of new MTD DAQ features. This allows the addition of new commands to be sent to the selected MTD controller.

GitLab Repository: [<xDAQ interface>¹](#)

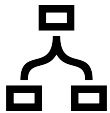
2.4.2.2 MTD Controller



The MTD Controller is typically deployed on the Serenity boards, where it is responsible for implementing the configuration received from the MTD Configurator. Additionally, it facilitates the transmission of monitoring data to the MTD collector by utilizing the event bus. The MTD Controller has two distinct interfaces: the Python interface and the xDAQ interface. The Python interface receives configurations and commands from the MTD configurator. Additionally, it periodically transmits monitoring data to the xDAQ interface of the MTD Controller. The xDAQ interface of the MTD Controller is responsible for transmitting the monitor data through the eventing bus using a previously determined topic name. This facilitates uninterrupted communication between the MTD controller and other components within the system. The Eventing Bus is a protocol specifically designed at CERN for data acquisition development applications. This protocol operates through a sender and a subscriber component. Data is transmitted by means of a pre-established topic name, whereby the recipients who have subscribed to that particular topic on the shared event bus can receive updates on the transmitted data.

GitLab Repository: [<Python interface>²](#) [<xDAQ interface>³](#)

2.4.2.3 MTD Collector



The MTD Collector implements an Eventing subscriber event protocol. As the service name implies, it gathers the monitoring, status, and log messages. Additionally, the system includes an Eventing publisher responsible for publishing the necessary messages to the WINCC OA SCADA system via the PSX DCS eventing bus as per the Detector Control System (DCS) requirements. Furthermore, the collected monitoring data is subjected to an anomaly detection model to proactively identify any operational anomalies, informing the monitoring shift operator of potential issues before they escalate beyond control. The collected monitoring, status, and log messages are stored in a dedicated monitoring database, which enables convenient access and retrieval of historical monitoring information, facilitating further analysis, reporting, and immediate visualization of monitoring data via the MTD DAQ GUI. The monitoring data enables operators to monitor system performance promptly and detect potential concerns. The stored data can also be utilized to conduct trend analysis and performance optimization, thereby improving the system's overall efficiency.

GitLab Repository: [<xDAQ interface>⁴](#)

2.4.2.4 MTD GUI

The MTD GUI front-end development was done in collaboration with Abdulrahman Almarzouqi, a UOB Summer student [40]. While I dedicated my efforts to backend development and related tasks, Abdulrahman played a crucial role in enhancing the front-end interface for the MTD DAQ Services using the Vue framework [41]; Abdulrahman's work on the frontend interface aimed to match the ease of interfacing new MTD. The web development backend provides DAQ services counterpart.

¹<https://gitlab.cern.ch/mtd-backend/daq/software/mtdconfigurator>

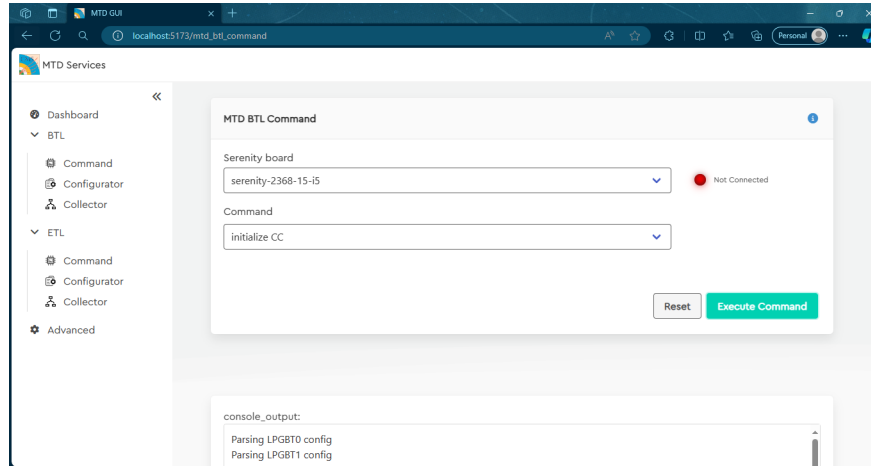
²<https://gitlab.cern.ch/mtd-backend/daq/software/mtdcontrollerpy>

³<https://gitlab.cern.ch/mtd-backend/daq/software/mtd>

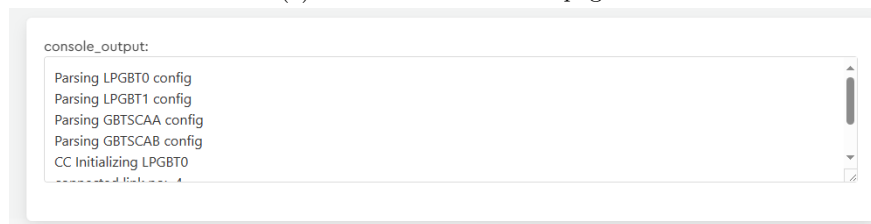
⁴<https://gitlab.cern.ch/mtd-backend/daq/software/mtd>

Commands page

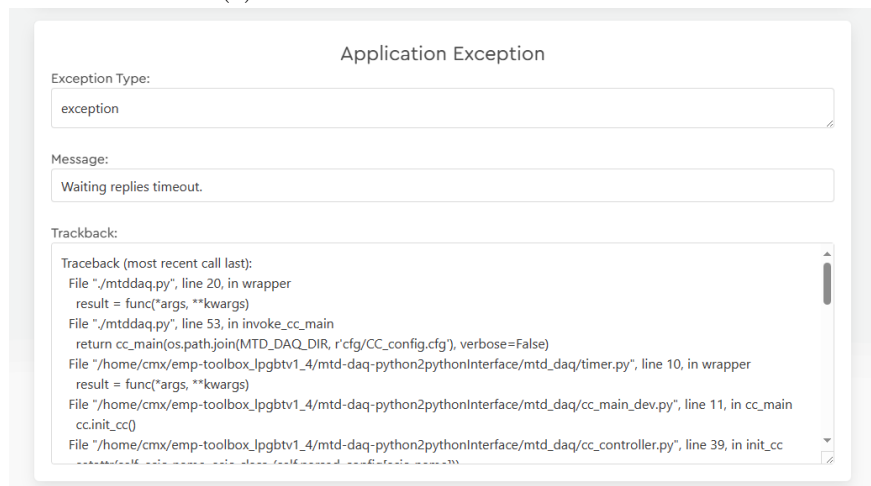
The commands page lets users choose an MTD DAQ sub-process to execute in the MTD controller and any necessary arguments (figure 2.10). The MTD controller initiates the execution of the sub-process by invoking the corresponding Python function. Subsequently, the program records the output displayed on the console and retrieves any specific information about errors or exceptions that occurred during the execution of the application. These details are then incorporated into the response (figure 2.10c).



(a) MTD BTL Command page



(b) MTD GUI Command selection results

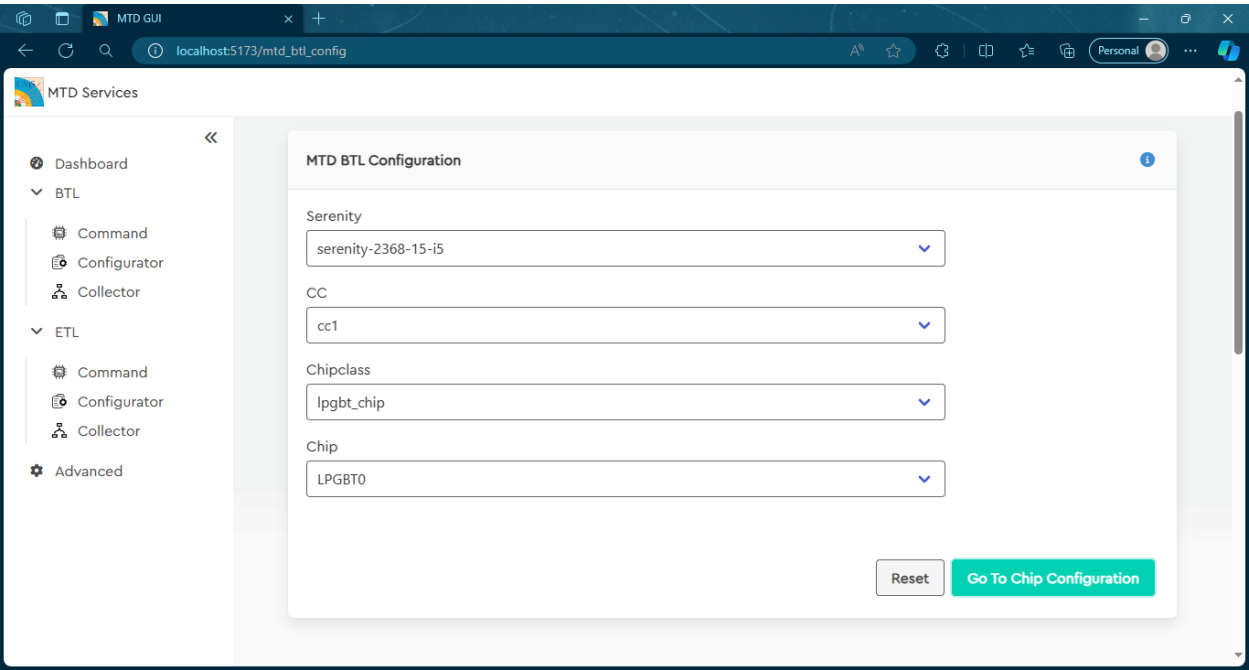


(c) MTD GUI Command selection results

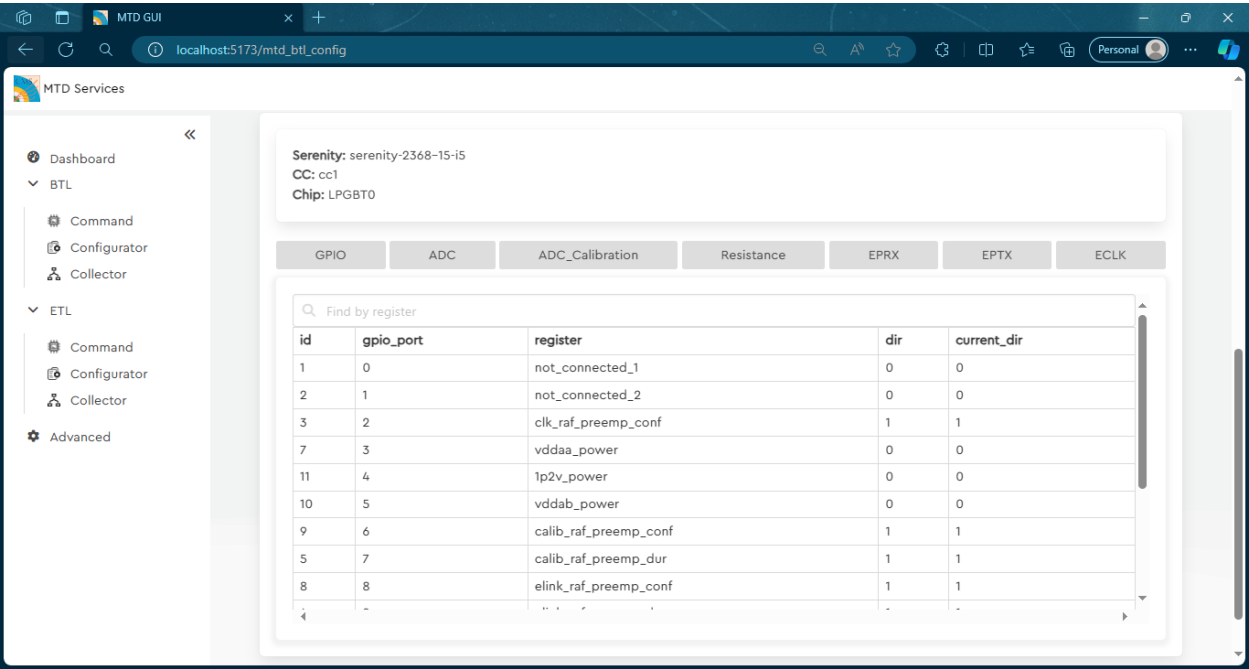
Figure 2.10: MTD GUI Command page

Configuration view page

The the MTD GUI Configurations page displays the configurations from the configurations database for the chip that you selected. Currently the database update is disabled where it is planned to add that feature after introducing login and MTD users where only the admin user are allowed to edit the configuration in the database.



(a) MTD GUI Configurations page



(b) MTD GUI Configurations view

MTD DAQ Anomaly Detector

This chapter constitutes the part of this work related to keeping the large array of sensors at the MTD operating normally. Hence, the goal is to maintain the integrity of the large array of sensors by detecting possible anomalies and triggering corrective measures before escalating possible problems. This is particularly important in light of the very harsh operation environment the sensors are subjected to, namely extreme temperature and radiation. It's important to note that the goal is not to address the resolution or correction of anomalous behaviors but rather to point out their occurrences. In this chapter, we will discuss the data acquisition (DAQ) anomaly detector design and model performance. Until the middle of this project, practical MTD data was unavailable, so simulated data were used to test and suggest ways to detect sensor anomalies. Subsequently, and towards the time of this thesis write-up, some partial sensor data became available and were crudely used to validate the simulation results.

In this chapter, we apply different AI techniques for anomaly detection, namely various AutoEncoder architectures, including Convolutional Neural Networks (CNNs), 2D Convolutional with Long Short-Term Memory (Conv2DLSTM), CNNs with U-net architecture, and Conv2DLSTM with U-net architecture. We have tested the recommended method's robustness against additive noise and investigated the wavelet denoising algorithm to mitigate this noise effect.

3.1 The anomaly definition

In monitoring the data-acquisition process, using the diagnosis sensors provides an early warning that the physical data-acquisition operation is less than optimal; if it is less than optimal, it is considered anomalous behavior. The slow-control electronics retrieve the readings from the diagnosis sensors specifically for use in this anomaly detector.

The anomalous data acquisition behaviors are only known after they occur. However, the most obvious anomaly occurs when the sensor readings exceed normal ranges, such as exceeding the specified thresholds. Sensor reading thresholding as an anomaly detector is widespread due to its simple implementation, where predefined thresholds are set to detect abnormal sensor readings. However, there have been examples of subtle anomalies, such as a region of sensors displaying elevated noise levels due to environmental factors, even though the readings do not exceed the sensor's normal range. A rudimentary implementation of thresholding for sensor readings cannot detect subtle anomalies. Hence, AutoEncoder models were used to pick up on these subtle anomalies. The anomalies will be examined in model comparisons to assess their effectiveness.

The expected anomalies are:

- **Out-of-range anomaly:** This is when any sensor readings exceed the designated range. This anomaly can be detected easily by setting a threshold on the expected ranges. As such, it's considered trivial. However, there are more subtle anomalies that machine learning techniques can detect.
- **Disconnected sensors anomaly:** Sensor reading can be disrupted for many reasons, from miscommunication to electronic faults.
- **Noisy regional sensors anomaly:** : This anomaly commonly occurs due to harsh environmental factors that impact certain areas of the sensor array, hence randomly corrupting the data. This anomaly shall be the focus of this work. (Figure 3.1).

Figures (3.1a, 3.1b, 3.1c) depict normal sensor windows for temperature, Silicon Photomultiplier (SiPm), and voltage/current respectively. Figures (3.1d, 3.1e, 3.1f) illustrate the same sensor windows but with the presence of Noisy regional sensors anomaly.

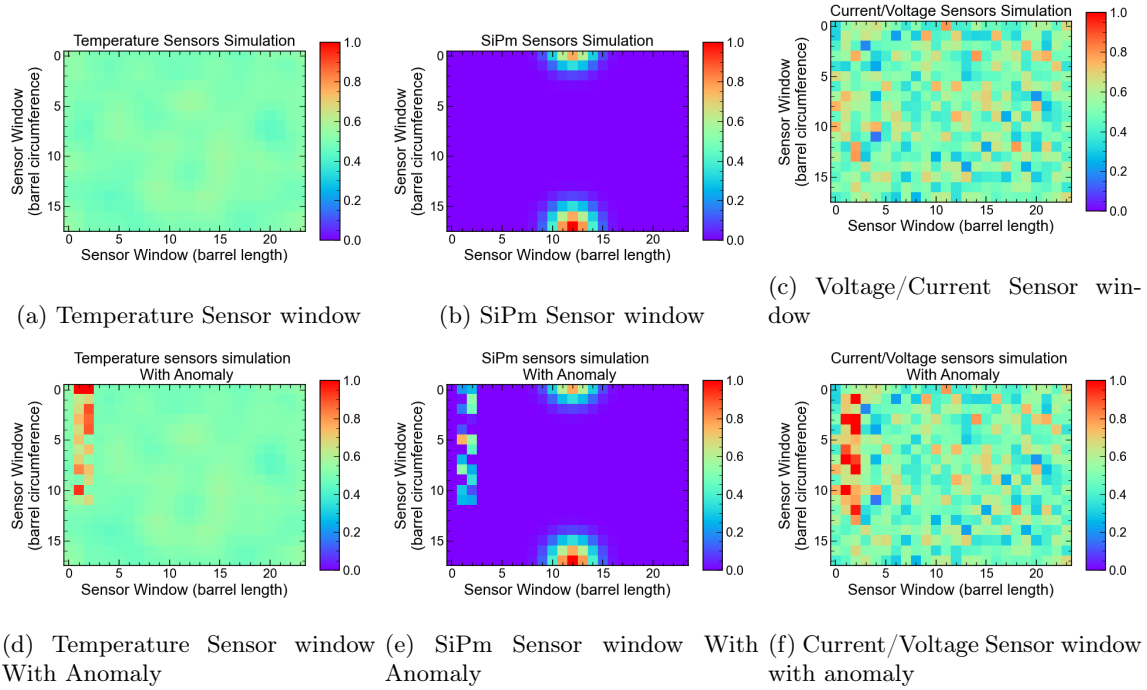


Figure 3.1: Initial Frame ($t_n = 0$) sensor window simulations with and without Anomaly

3.2 Anomaly Detection Algorithms

3.2.1 Anomaly Detection methods selection criteria

The anomalous behavior in data acquisition can not be anticipated as it can mostly be recognized after it has occurred. On the other hand, normal operation is available and plentiful. Hence, the Semi-supervised (one-class classification) models are ideal for our use case since they are only trained on normal operation datasets. These models can detect anomalous instances by computing the dissimilarity of the dataset representing normal operations. The one-class classification model's output is ultimately binary, categorizing it as either an "**Anomaly**" or "**Normal**" labeled positive and negative, respectively.

The dataset will be significantly large due to the substantial number of sensors (around 66 thousand) and the multiplication by the time dimension. Consequently, Training the models in multiple batches is the only feasible option. Currently, it is not possible to train an isolated forest using batches. While SVM has made advancements in incremental learning [42]. However, it is exclusively designed for multi-class classification and cannot be used for one-class classification. Therefore, employing classification methods like isolation forest and one-class SVM is not practical in our particular situation.

Exploring dimensionality reduction techniques such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). PCA offers several algorithms that facilitate incremental learning [43][44]. The SVD incremental learning algorithm is discussed in the paper with the reference [45]. Nevertheless, it is customary to perform data vectorization prior to applying PCA, resulting in the loss of specific details such as the spatial positioning within the sensor window and the temporal sequence. ConvLSTM2D from TensorFlow is recommended for temporal dimensioning sensor window. Since TensorFlow layers are well-tested and stable, they are preferred over other 2D RNNs not included in The TensorFlow package.

The sensors' operational range is subject to variation depending on the prevailing conditions in the cavern environment. This is factored in the pre-processing, where the data is re-scaled such that the maximum range maps to 1 and the minimum range maps to 0. After the pre-processing, all the values are expected to be between $[0, 1]$ if the values are outside the range, by definition, are considered out-of-range anomalies.

As such, the Anomaly detection selection criteria are as follows:

- Semi-supervised (One-class Class classifier).
- Supports multi-batching model fitting.
- Possesses noise robustness in view of the harsh operating environment.
- Supports Time Series data.
- The input sensor's normal operating range is subject to change.
 - The model accepts values between 0 and 1.
 - The dataset is pre-processed accordingly before the model input (section 3.2.2).

As such, *AutoEnocders* fit the requirements perfectly.

3.2.2 Pre-processing: Re-scaling and Denoising

3.2.2.1 Re-scaling

Sensor range is defined for each sensor depending on the current expected environmental condition. As such, the re-scaling of the sensor data reading is applied for each sensor accordingly. The re-scale is as follows:-

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (3.1)$$

Let X_{min} and X_{max} be the minimum and maximum range for the sensor.

It is important to note that X_{min} and X_{max} represent known values. Any data points that fall outside this range are automatically classified as anomalies. Data points within the range are potentially considered anomalies and are therefore inputted into the anomaly detector model. As such the anomaly model will always have data within range $\in [0, 1]$.

3.2.2.2 Denoising

Wavelet denoising is the best option for eliminating white Gaussian noise, as supported by previous studies [23]. Denoising shall be applied to the anomaly detection method of choice as will be presented in section 4.1 in the context of testing its robustness and alleviating the noise effect. In selecting the wavelet function, mode, and threshold for denoising my specific dataset, I took an empirical approach and utilized a Genetic algorithm with the PyGAD tool [46]. The "Gene" represents the chosen wavelet function, mode, and threshold. In contrast, the fitness function is calculated as the inverse of the mean squared error (MSE) between a sample of the original dataset and a version with added noise that is then denoised.

The gene parameters ranges are as shown in Table 3.1 with the best solution after 10 Generations after the solution is saturated for 5 generations. The Genetic algorithm (GA) is configured as follows:-

- Number of Generations (**num_generations**): 50
- Number of Parents Mating (**num_parents_mating**): 10
- Fitness Function (**fitness_func**): the Inverse of the mean squared error (MSE) between a sample of the original dataset and a version with added noise that is then denoised.
- Solutions Per Population (**sol_per_pop**): 50
- Gene Space: Table 3.1
- Crossover Type (**crossover_type**): Uses "single_point" crossover, a method where a single crossover point on both parent's genome is selected and the genes are swapped to produce new offspring.
- Mutation Type (**mutation_type**): Uses "random" mutation, where genes of individuals can randomly mutate, potentially exploring new areas of the solution space
- Stop Criteria (**stop_criteria**): The algorithm will stop if the best solution's fitness doesn't improve for 5 consecutive generations ("saturate_5").
- Allow Duplicate Genes (**allow_duplicate_genes**): Set to **False** to ensure all genes within a single solution are unique.

Table 3.1: Wavelet parameters selection ranges (Gene Space) and solution

Gene	Data type	range	GA Solution
Threshold	float	$\in [10^{-9}, 1]$	0.9938780667668272
wavelet	string	pywt.wavelist(kind='discrete')	<i>db1</i>
mode	string	['soft', 'hard', 'garrote', 'greater', 'less']	<i>soft</i>

Table 3.2: Wavelet Functions from pywt.wavelist(kind='discrete')

Type	Wavelets
bior	bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4, bior5.5, bior6.8
coif	coif1, coif2, coif3, coif4, coif5, coif6, coif7, coif8, coif9, coif10, coif11, coif12, coif13, coif14, coif15, coif16, coif17
db	db1, db2, db3, db4, db5, db6, db7, db8, db9, db10, db11, db12, db13, db14, db15, db16, db17, db18, db19, db20, db21, db22, db23, db24, db25, db26, db27, db28, db29, db30, db31, db32, db33, db34, db35, db36, db37, db38
dmey	dmey
haar	haar
rbio	rbio1.1, rbio1.3, rbio1.5, rbio2.2, rbio2.4, rbio2.6, rbio2.8, rbio3.1, rbio3.3, rbio3.5, rbio3.7, rbio3.9, rbio4.4, rbio5.5, rbio6.8
sym	sym2, sym3, sym4, sym5, sym6, sym7, sym8, sym9, sym10, sym11, sym12, sym13, sym14, sym15, sym16, sym17, sym18, sym19, sym20

The genetic algorithm will optimize the fitness function to find the wavelet function, mode, and threshold combination that minimizes the MSE where this combination is selected. This approach allows for a systematic and data-driven method to determine the best parameters for denoising in a way that is tailored to my data's specific characteristics. Denoising is applied to individual sensor windows in the event data [47].

3.2.3 Auto-encoder models

3.2.3.1 ConvReshape Layer

During the MTD development phase, the input that include a combination of sensors and readout units will change when adding new readout units and sensors. Therefore, the model needs to be adaptable to new input shapes in the sensor window and sensor dimensions represented in the last 3 dimensions. The ConvShaping layer has been developed to reshape the layer input into a suitable output using convolution, allowing for flexibility in both input and output shapes. Moreover, the remaining components of the AutoEncoder model can remain unaltered. Furthermore, by applying convolution to the last three dimensions (which represent the sensor window and sensors). The ConvShaping layer can extract features instead of simply reshaping with no regard for the feature context through up or down-sampling. The ConvShaping layer takes input data with dimensions $(b, 30, \alpha, \beta, \gamma)$ and outputs data with dimensions $(b, 30, A, B, C)$ as illustrated in figure 3.2a. The variable "b" denotes the batch size, whereas the value 30 corresponds to the number of data time frames. The recording process captures each time frame at a frequency of one frame per second, continuing for a duration of 30 seconds. In the model input, the values of (α, β) represent the sensor window, and γ represents the sensor dimension. On the other hand, (A, B, C) are determined based on the AutoEncoder used. At the AutoEncoder model output, the sensor window will be represented by the coordinates (A, B) , and the dimensions of the sensors will be C . The values of α, β, γ will be determined based on the AutoEncoder being used.

3.2.3.2 AutoEncoders architectures

This thesis compares the efficiency of four AutoEncoder architectures (LSTM-AE, CNN-AE, U-net CNN, and U-net LSTM) to detect anomalies using TensorFlow framework [48]. This test will mostly look at The accuracy of the system can find anomalies in the time series dataset of each architecture to ascertain the most appropriate one for anomaly identification in MTD DAQ operations.

The four AutoEncoder architectures are the following.

- **CNN-AE:** The CNN-AE is a standard AutoEncoder that captures temporal patterns in the data using convolution (Conv3D). Additionally, the use of convolutional layers in the CNN-AE allows for feature extraction and dimensionality reduction. This makes it suitable for capturing complex patterns in the data while reducing computational complexity. (figure 3.2b)
- **LSTM-AE:** The LSTM-AE model has a structure similar to the CNN-AE model. However, ConvLSTM2D was used in place of Conv3D layers with sequence-to-sequence since the throughout the model the input and out are the entire sequence (not processing one time frame at a time). Since the data is three dimensional (2D sensor window and sensors) over time, ConvLSTM2D is a good fit to handle the temporal aspect of the data. 3.2c
- **U-net CNN:** The U-shaped design of the U-net network architecture, which includes symmetric encoder and decoder pathways linked by skip connections, increases efficiency in detecting subtle features in the data. The U-net architecture an excellent at extracting relevant features. (figure 3.3a)
- **U-net LSTM:** The U-net LSTM model has a structure similar to the U-net CNN model. However, LSTM-AE uses ConvLSTM2D in place of Conv3D layers. Additionally, incorporating LSTM layers allows the model to capture long-term dependencies in sequential data. Note that ConvLSTM2D are used with sequence-to-sequence since the throughout the model the input and out are the entire sequence (not processing one time frame at a time). (figure 3.3b)

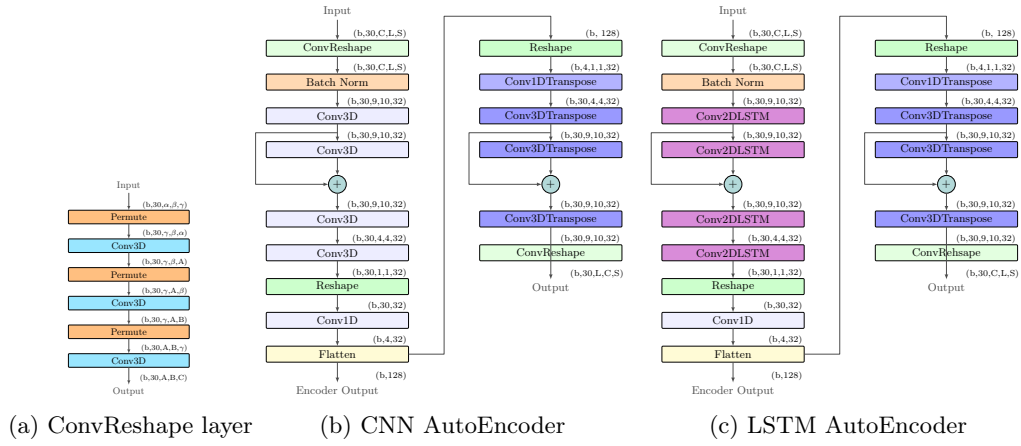


Figure 3.2: AutoEncoders

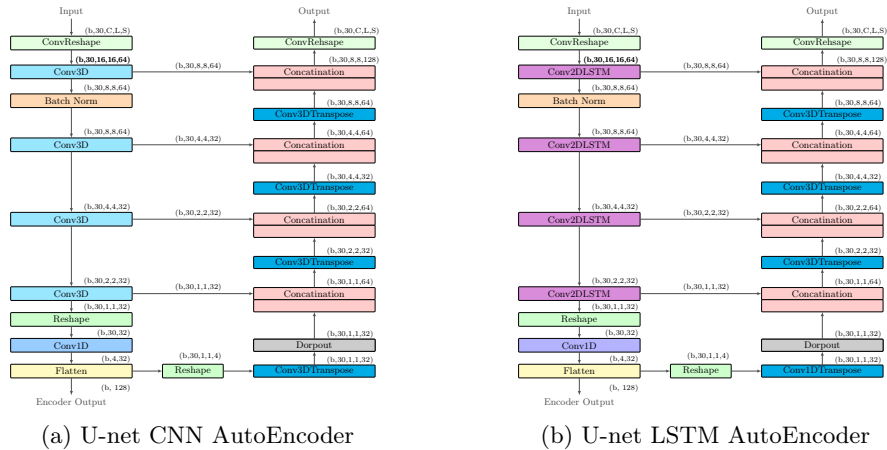


Figure 3.3: U-net AutoEncoders

3.2.4 One-Class classification methods

3.2.4.1 AutoEncoder's MSE

This one-class classification classifies anomalies if the MSE exceeds the MSE threshold. Considering normal distribution $N \sim (\mu = 0, \sigma = 1)$, the MSE less than the threshold with 95% confidence is the area under the curve when $P(Z < z) = 0.95$, which is ($z = 1.645$). Thus the threshold is:

$$\text{MSE threshold} = \mu + 1.645\sigma \quad (3.2)$$

Algorithm 1 MSE Autoencoder One-class Classification

Require: $AE \leftarrow \text{AutoEncoder}$

```

1: procedure TRAIN MSE AUTOENCODER CLASSIFIER( $\{\{\mathbf{X}\}_{1:m}\}_{1:n}, AE$ )           ▷ Denote dataset
    $\{x_1, x_2, \dots, x_{n-1}, x_n\}$  as  $\{X\}_{1:n}$ 
2:   Train AutoEncoder  $AE$  with  $\{\mathbf{X}\}_{1:n}$ 
3:    $\mu, \sigma \leftarrow 0, 0$                                            ▷ Initialize mean and std. dev.
4:   for  $\{\mathbf{X}\}_{1:m} \in \{\mathbf{X}_{1:m}\}_{1:n}$  do
5:      $\{\mathbf{Y}\}_{1:c} \leftarrow \{\text{MSE}(AE(X), X) | X \in \{\mathbf{X}\}_{1:m}\}$ 
6:      $\mu, \sigma \leftarrow \text{UpdateStats}(\mu, \sigma, \{\mathbf{Y}\}_{1:c})$            ▷ Section A.2
7:   end for
8:   threshold  $\leftarrow \mu + z_\alpha \sigma$                                ▷ Set threshold (e.g.,  $z_\alpha = 1.645$  for 95% confidence)
9:                                           ▷ equation 3.2
10:  return threshold
11: end procedure
12: procedure MSE AUTOENCODER ONE-CLASS CLASSIFICATION( $\mathbf{x}, AE, \text{threshold}$ )
13:   $y_{pred} \leftarrow AE(\mathbf{x})$ 
14:   $MSE_{pred} \leftarrow \text{MSE}(AE(\mathbf{x}), \mathbf{x})$                                ▷ Calculate MSE for input sample
15:  if  $MSE_{pred} > \text{threshold}$  then return Anomaly                     ▷ Sample is anomalous
16:  elsereturn Normal                                                 ▷ Sample is normal
17:  end if
18: end procedure

```

Figure 3.4: MSE Autoencoder One-class Classification Algorithm

3.2.4.2 k-means one-classification

The one-class classification presented herein uses the k-means clustering algorithm. k-means used is multi-batched k-means clustering algorithm implemented in scikitlean as MiniBatchKMeans [49]. The anomaly is detected when the Mahalanobis distance exceeds the highest threshold value. Initially, the classifier undertakes the fitting of a k-means clustering classifier, wherein each batch of latent space vectors of 128 dimensions ($\mathbb{R}^{128}$) undergoes partial fitting iteratively. Subsequently, it computes essential cluster statistics such as covariance matrices and vector means for each cluster, followed by its iterative updates based on the latent space vectors. Further, the algorithm computes the maximum Mahalanobis distance for each cluster, a pivotal measure in discerning the outliers within the data. This process entails the iterative calculation of the Mahalanobis distance for each vector relative to its cluster's mean and covariance matrix. These steps culminate by establishing a robust k-means one-class classification model that delineates anomalies within the data domain. Multi-batching is implemented in K-means via efficiently updating the scalar and vector statistics, namely the mean and covariance. For the sake of compactness, the derivations of the updating algorithm are given in Appendix A.3

Algorithm 2 K-means One-class Classification

```

1: procedure TRAIN K-MEANS ONE-CLASS CLASSIFIER( $\{\{\vec{x}\}_{1:n}\}_{1:b}, c$ )  $\triangleright c$ : Number of clusters  $\triangleright \vec{x}$  is
   with 'k' dimensions  $\triangleright$  Denote dataset of  $\{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_{n-1}, \vec{x}_n\}$  as  $\{\vec{x}\}_{1:n}$ 
2:    $Kmeans \leftarrow$  Initialize K-means Classifier
3:   for  $\{\vec{x}\}_{0:n} \in \{\{\vec{x}\}_{0:n}\}_{1:b}$  do
4:      $Kmeans.partial\_fit(\{\vec{x}\}_{0:m})$   $\triangleright$  Fit K-means in batches
5:   end for
6:    $\{C\}_{1:c} \leftarrow \{I_k\}_{1:c}$   $\triangleright$  Initialize cluster Covariance
7:    $\{\vec{\mu}\}_{1:c} \leftarrow \{0_{1 \times k}\}_{1:c}$   $\triangleright$  Initialize cluster mean
8:   for  $\{\vec{x}\}_{1:n} \in \{\{\vec{x}\}_{0:n}\}_{1:b}$  do
9:     for  $m \in [1, 2, \dots, c]$  do  $\triangleright$  iterate though clusters
10:       $\{\vec{y}\}_{1:p} \leftarrow \{\vec{z} \in \{\vec{x}\}_{1:n} | m == Kmeans.predict(\vec{z})\}$ 
11:       $C_m, \vec{\mu}_m \leftarrow UpdateVectorStats(C_m, \vec{\mu}_m, \{\vec{y}\}_{1:p})$ 
12:    end for
13:  end for
14:   $\{d\}_{1:c} \leftarrow \{0\}_{1:c}$   $\triangleright$  Initialize max Mahalanobis distances
15:  for  $\{\vec{x}\}_{1:n} \in \{\{\vec{x}\}_{0:n}\}_{1:b}$  do
16:    for  $m \in [1, 2, \dots, c]$  do  $\triangleright$  iterate though clusters
17:       $\{\vec{y}\}_{1:p} \leftarrow \{\vec{z} \in \{\vec{x}\}_{1:n} | m == Kmeans.predict(\vec{z})\}$ 
18:       $d_m \leftarrow \max(\{d_m\} \cup \{MahalanobisDistance(\vec{z}, C_m, \vec{\mu}_m) | \vec{z} \in \{\vec{y}\}_{1:p}\})$ 
19:    end for
20:  end for
21:  return  $Kmeans, \{C\}_{1:c}, \{\vec{\mu}\}_{1:c}, \{d\}_{1:c}$ 
22: end procedure
23:
24: procedure K-MEANS ONE-CLASS CLASSIFIER( $\vec{x}, \{C\}_{1:c}, \{\vec{\mu}\}_{1:c}, \{d\}_{1:c}, \alpha$ )
25:    $k \leftarrow Kmeans.predict(\vec{x})$   $\triangleright$  Get nearest cluster for  $\vec{x}$ 
26:    $\vec{\mu}, C, d_{max} \leftarrow \vec{\mu}_k, C_k, d_k$   $\triangleright$  Get cluster stats
27:    $d \leftarrow MahalanobisDistance(\vec{x}, \vec{\mu}, C)$   $\triangleright$  Calculate Mahalanobis distance
28:   if  $d > d_{max}(1 + \alpha)$  then return Anomaly  $\triangleright \alpha$ : Tolerance factor
29:   else return Normal
30:   end if
31: end procedure

```

Figure 3.5: K-means One-class Classification Algorithm

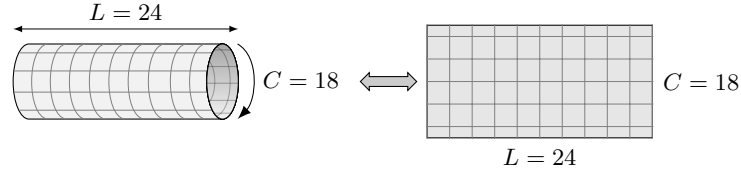
3.3 Anomaly Detection Simulation

At the time of writing this thesis, real data is still being gathered from the massive MTD sensor array. Hence, simulated data matching the real ones shall be used to assess the proposed algorithms in this work

3.3.1 Dataset

The data are in the form $(b \times t_n \times C \times L \times S)$. Where $(C \times L)$ is the distribution of the BTL readout units, also known as Concentrator card (CC), along the barrel circumference C and length L . Henceforth, this data slice will be known as the sensor window (figure 3.6), S is the sensor, and t_n is the time frame instance. Finally, b is the batch size of the data input into the autoencoder. The AutoEncoders are trained using 10 batches, each containing batch size of 100 samples ($b = 100$), resulting in a total dataset size of 1,000.

The complete count of the sensors is 78, details of the sensors are in table 2.2 the sensor window is (18×24) , the data is simulating data recording of 1 frame per second in period of half a minute totaling 30 time frames, thus the simulated dataset elements of shape $(30 \times 18 \times 24 \times 78)$. The data are consistently constrained within the $[0,1]$ interval Since pre-processing sensor ranges are assumed.

Figure 3.6: BTL Readout Units representation as a sensor window of $(C \times L)$

The simulation of SiPM sensor activity may involve the passage of many MIP particles within a single event data. The occurrence of MIP heat signatures is determined by the following probabilities: 0% for a count of 10, 40% for a count of 1, and 50% for a count of 2.

Noise level	SNR (db)
low	17
medium	11
high	5

Table 3.3: SNR levels used in the robustness tests

To evaluate model robustness, white noise is introduced at three levels as listed in Table 3.3 to the simulated noiseless data. Where the Signal-to-Noise Ratio (SNR) is defined by:

$$\text{SNR} = 10 \log_{10} \left(\frac{P_s}{P_n} \right) \quad (3.3)$$

where P_s denotes the average sensor signal power (squared average sensor value), and P_n denotes the noise power (noise variance).

3.3.2 Activity simulation

The activity simulation generates the sensor window time sequence. Hence, the data slice of the simulation is $(t_n CL)$. There are three simulation types for three kinds of sensor window sequences.

- Temperature sensors (figure 3.1a).
- SiPM Temperature sensors (Figure 3.1b).
- Current/Voltage sensors (Figure 3.1c).

The temperature sensor windows are expected to exhibit a certain degree of correlation with the adjacent sensors such that the temperature a continuous two-dimensional function, a characteristic of the heat function (figure 3.1a). Let Denote the sensor window $(t_n \times C \times L)$ as $S_{c,l}^n$. To achieve this effect at the initial sensor window ($t_n = 0$), first $S_{i,j}^0$ is calculated as in eq 3.4 then averaging neighbors using convolution as in eq 3.5 3 times.

$$S_{i,j}^0 = U(0.4, 0.6) + N(0.0, 0.1) \quad (3.4)$$

$$S_{i,j}^0 = S_{i,j}^0 * \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \quad (3.5)$$

where

$$U(a, b): \text{Uniform random variable } \in [a, b] \quad (3.6)$$

$N(\mu, \sigma)$: Normally distributed random variable with mean μ and standard deviation σ

$*$: Convolution operator

However, sensor windows for the SiPM temperature sensor's initial state ($t_n = 0$) are simulated using 2D Gaussian (eq 3.7) to approximate the effect of passing MIP particle:

$$S_{x,y}^0 = \frac{1}{2\pi\sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2}[x-x_0, y-y_0]\Sigma^{-1}\begin{bmatrix} x-x_0 \\ y-y_0 \end{bmatrix}\right) \quad (3.7)$$

where:

(x_0, y_0) : Location of passing MIP particle on the sensor grid.

Σ : The Covariance matrix determines the heat impact and direction on the sensor window grid in this simulation.

After the temperature sensors and SiPM temperature initial sensor windows are calculated (figure 3.1b), the subsequent sensor window frames are calculated using the heat equation. The distribution of heat follows the heat equation which is as follows using partial differential equation notation [50]: $u_t = \alpha(u_{xx} + u_{yy})$.

The calculation of temperature sensor window frames involves the utilization of a discrete form of the Heat equation in an iterative manner where the subsequent sensor window frame depend on the current sensor window frame using eq 3.8. For the sake of conciseness, the specific derivation of the discrete form of the heat equation can be found in Appendix A.1.

$$u_{x,y}^{n+1} \approx u_{x,y}^n + (\Delta t)\alpha \left(\frac{u_{x+2,y}^n - 2u_{x+1,y}^n + u_{x,y}^n}{(\Delta x)^2} + \frac{u_{x,y+2}^n - 2u_{x,y+1}^n + u_{x,y}^n}{(\Delta y)^2} \right) \quad (3.8)$$

The boundary comply the cylindrical geometry of the sensor window (figure 3.6) :

$$u_{i+1,y}^n = u_{0,y}^n \quad (3.9)$$

$$u_{x,j+1}^n = u_{x,j-1}^n \quad (3.10)$$

$$u_{x,-1}^n = u_{x,1}^n$$

Where (i,j) are the last index, respectively.

The cooling plate's material is Aluminum. Hence, the thermal diffusivity used in the simulation is $\alpha = (8.4 \pm 0.3) \times 10^{-5} \text{cm}^2 \text{s}^{-1}$ [51]. The MTD BTL barrel surface area is $\sim 38 \text{m}^2$ and radius of $1168 \pm 20 \text{mm}$ [52] then:

$$\Delta x = \frac{\text{Circumference}}{18} = \frac{2\pi(1.168\text{m})}{18} \approx 0.40771\text{m} \quad (3.11)$$

$$\Delta y = \frac{\text{length}}{24} = \frac{\frac{\text{surface area}}{\text{Circumference}}}{24} = \frac{\frac{38\text{m}^2}{2\pi(1.168\text{m})}}{24} \approx 0.21575\text{m} \quad (3.12)$$

$$\Delta t = 1\text{s} \quad (3.13)$$

The voltage and current sensor windows (figure 3.1c) do not have a dependency on neighboring sensors. Therefore, the initial sensor windows ($t_n = 0$) are simulated using random values as in eq 3.4. Then, the subsequent sensor windows are calculated using random walks with small steps by adding white noise of step of $\sigma^2 = 0.01$ as in eq 3.14.

$$S_{i,j}^{n+1} = S_{i,j}^n + N(0, 0.01) \quad (3.14)$$

3.3.3 Anomaly simulation

The anomaly that will be simulated is corrupted local sensors since Out-of-range and disconnected sensors don't require machine learning to be detected. It is simulated by defining a 2×12 rectangle randomly placed within some sensor windows. This rectangle displays elevated levels of noise in comparison to the surrounding environment. The specifications were obtained through consultation with the MTD DAQ Team. These specifications serve as a reliable metric for conducting performance checks on Anomaly Detection models, as they identify anomalous behavior based on their expert opinion.

3.4 Real Data

As previously mentioned, only partial sensor readings were acquired due to the ongoing development phase of the MTD DAQ. These readings consist of 23 temperature sensors from one BTL readout unit, also known as concentrator cards (CC). Therefore, the sensor window is a single CC (1×1), capturing sequences of 30 frames resulting in a dataset with elements of shape $(30 \times 1 \times 1 \times 23)$. Temperature sensors that were included in the partial dataset are listed in Table 3.4.

Sensor	Count
CC board Temperature	3
PCC A Temperature	2
PCC B Temperature	2
FE1 SIPM Temperature	2
FE2 SIPM Temperature	2
FE5 SIPM Temperature	2
FE6 SIPM Temperature	2
FE7 SIPM Temperature	2
FE8 SIPM Temperature	2
FE11 SIPM Temperature	2
FE12 SIPM Temperature	2
Total	23

Table 3.4: Sesnors that were included in real data

When the sensor reading was taken, the equipment was at room temperature. As such, the temperature range was set to between $[5, 45]C^{\circ}$. Hence, when rescaling was done using equation 3.1 all the data were within $[0, 1]$. As with any sensor reading, the presence of white noise is expected. The white noise measurement was $N(0, 0.0115)$, which is relatively low since the sensors are new and have not been subjected to a harsh environment yet.

No anomalous measurements have been recorded as of now hence simulated anomalies were also used to test the algorithms on the real data. The data sequences were divided into three datasets to evaluate the anomaly detection model: normal training, normal testing, and anomalous testing. The ratio used for this division was 80:10:10.

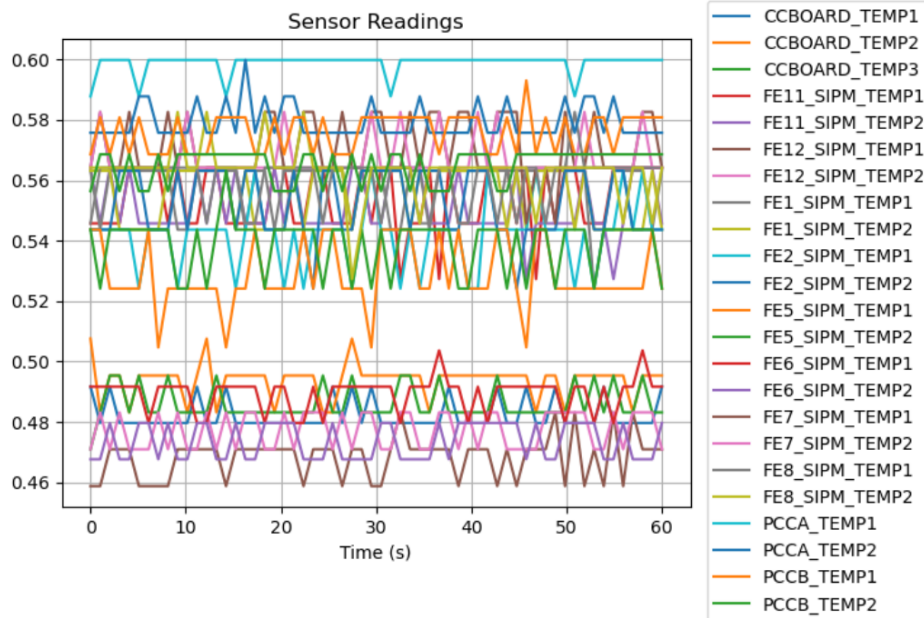


Figure 3.7: One-minute excerpt of sensor reading

Results and recommendations

This chapter will display the anomaly model's performance in detecting anomalies within the simulated and partial datasets. The recommendations will be based on how effectively each model accurately identifies anomalies. The best model will be selected based on its ability to detect anomalies accurately. The overall anomaly detection consists of denoising pre-processing, using the neural network, and finally, using the one-class classifier.

$$\left\{ \begin{array}{l} \text{Denoising applied} \\ \text{Denoising not applied} \end{array} \right\} \left\{ \begin{array}{l} \text{CNN-AE} \\ \text{LSTM-AE} \\ \text{U-net CNN} \\ \text{U-net LSTM} \end{array} \right\} \left\{ \begin{array}{l} \text{MSE} \\ \text{k-means OC} \end{array} \right\} \quad (4.1)$$

4.1 Results On Simulated Data

The above autoencoder architectures were tested with the simulated data set presented in Section 3.3.3. Tables in figure 4.1 summarize the results using k-means clustering one class classification on the four proposed autoencoders architectures. Similarly, Tables in figure 4.2 provide the performance result using MSE one-class classifications for the same four autoencoder architectures. Superior performance of 96.2 % was attained with **U-net LSTM autoencoder** with **MSE one-class classifications**.

The performance of k-means one-class classification was poor because the clusters of anomaly and normal latent space vectors were not sufficiently compact to allow for differentiation. Additional investigation is required to examine the effectiveness of this algorithm using a publically accessible dataset and determine the conditions of its appropriate usage.

Further tests were performed to assess the robustness of the anomaly detection model by adding white noise to the data set with different Signal-to-Noise Ratios (5dB, 11dB, and 17dB). Table 10 depicts the results showing a marginal degradation of the performance (only 1.1 %) with the data quality reduced from 17 to 5 dB, hence validating the denoising capabilities of the autoencoders.

Finally, to further mitigate the noise effect, the Wavelet denoising [24] was applied to the data set before feeding it to the algorithms. Table 11 illustrates the results showing a degradation in performance for medium to low SNR cases where the anomalies were caused to be hidden hence missed by the algorithm.

AutoEnocder			Accuracy	AutoEnocder			Accuracy	AutoEnocder			Accuracy	AutoEnocder			Accuracy				
CNN			87.77%		U-net CNN			81.68%		U-net LSTM			68.77%		LSTM			50.18%	
Actual	Positive	46.14%	3.86%	Actual	Positive	46.18%	3.81%	Actual	Positive	48.81%	1.18%	Actual	Positive	0.32%	49.68%				
	Negative	11.36%	38.64%		Negative	14.50%	35.50%		Negative	30.05%	19.96%		Negative	0.14%	49.86%				
		Positive	Negative			Positive	Negative			Positive	Negative			Positive	Negative				
Predicted				Predicted				Predicted				Predicted							
(a) CNN AutoEncoder				(b) U-net CNN AutoEncoder				(c) U-net LSTM AutoEncoder				(d) LSTM AutoEncoder							

Figure 4.1: Anomaly detection results using K-means one-class classification

AutoEncoder			Accuracy	AutoEncoder			Accuracy	AutoEncoder			Accuracy	AutoEncoder			Accuracy				
U-net LSTM			96.20%		U-net CNN			81.68%		LSTM			89.25%		CNN			50.15%	
Actual	Positive	46.20%	3.80%	Actual	Positive	45.90%	4.10%	Actual	Positive	42.95%	7.05%	Actual	Positive	0.15%	49.85%				
	Negative	00.00%	50.00%		Negative	00.00%	46.30%		Negative	3.7%	46.30%		Negative	00.00%	50.00%				
		Positive	Negative			Positive	Negative			Positive	Negative			Positive	Negative				
	Predicted		Predicted		Predicted		Predicted		Predicted		Predicted								
(a) U-net LSTM AutoEncoder				(b) U-net CNN AutoEncoder				(c) LSTM AutoEncoder				(d) CNN AutoEncoder							

Figure 4.2: Anomaly detection results using MSE one-class classification

SNR (db)			Accuracy	SNR (db)			Accuracy	SNR (db)			Accuracy
5			94.25%	11			94.45%	17			95.35%
Actual	Positive	44.25%	5.75%	Actual	Positive	44.45%	5.55%	Actual	Positive	45.35%	4.65%
	Negative	00.00%	50.00%		Negative	00.00%	50.00%		Negative	00.00%	50.00%
		Positive	Negative				Positive			Negative	
Predicted				Predicted				Predicted			
(a) SNR 5 db				(b) SNR 11 db				(c) SNR 17 db			

Figure 4.3: Noise Robustness test results for LSTMUnet-MSE

SNR (db)			Accuracy	SNR (db)			Accuracy	SNR (db)			Accuracy
5			71.75%	11			80.30%	17			95.25%
Actual	Positive	21.75%	28.25%	Actual	Positive	30.30%	19.70%	Actual	Positive	45.25%	4.75%
	Negative	00.00%	50.00%		Negative	00.00%	50.00%		Negative	00.00%	50.00%
		Positive	Negative				Positive			Negative	
Predicted				Predicted				Predicted			
(a) SNR 5 db				(b) SNR 11 db				(c) SNR 17 db			

Figure 4.4: Noise Robustness test results for LSTMUnet-MSE with denoise preprocessing

The Denoising preprocessing hinders the overall robustness of the anomaly detection model by obscuring the anomaly. Furthermore, the autoencoder has already demonstrated its resilience to noise without requiring denoising preprocessing.

4.2 Results On Real Data

The anomaly detection has been performed using actual data as described in section 3.4 on all four autoencoder architectures and the k-means and MSE one-class classification methods. Similar to the simulation results, The MSE one-class classification has shown a significant improvement compared to the k-means one-class classification as shown in Figures 4.5 and 4.6a. The CNN autoencoder with MSE achieved the highest accuracy, 94.59%. The U-net LSTM model. The optimal model selected from the simulation achieved a comparable accuracy of 94.49%.

AutoEnocder Accuracy			AutoEnocder Accuracy			AutoEnocder Accuracy			AutoEnocder Accuracy		
LSTM			88.53%	CNN			11.47%	U-net LSTM			11.47%
Actual	Positive	0.00%	11.47%	Actual	Positive	11.47%	0.00%	Actual	Positive	11.47%	0.00%
	Negative	0.00%	88.53%		Negative	88.53%	0.00%		Negative	88.53%	0.00%
		Positive	Negative			Positive	Negative			Positive	Negative
Predicted				Predicted				Predicted			
(a) LSTM AutoEncoder - on real data				(b) CNN AutoEncoder - on real data				(c) U-net LSTM AutoEncoder - on real data			
								(d) U-net CNN AutoEncoder - on real data			

Figure 4.5: Anomaly detection results using k-means one-class classification on real data

AutoEnocder		Accuracy	
CNN		94.59%	
Actual	Positive	11.15%	0.00%
	Negative	5.40%	83.44%
		Positive	Negative
		Predicted	

(a) CNN AutoEncoder - on real data

AutoEnocder		Accuracy	
LSTM		94.56%	
Actual	Positive	11.15%	0.00%
	Negative	5.44%	83.40%
		Positive	Negative
		Predicted	

(b) LSTM AutoEncoder - on real data

AutoEnocder		Accuracy	
U-net LSTM		94.49%	
Actual	Positive	11.15%	0.00%
	Negative	5.51%	83.34%
		Positive	Negative
		Predicted	

(c) U-net LSTM AutoEncoder - on real data

AutoEnocder		Accuracy	
U-net CNN		94.05%	
Actual	Positive	11.15%	0.00%
	Negative	5.95%	82.89%
		Positive	Negative
		Predicted	

(d) U-net CNN AutoEncoder - on real data

Figure 4.6: Anomaly detection results using MSE one-class classification on real data

Conclusion

5.1 Summary

In this work, we investigated the application of Artificial Intelligence (AI) and Signal Processing (SP) techniques in the context of Sensor Fusion applied to the high-energy physics experiment in the CMS experiment at CERN, being one of the megaprojects that are being carried out at the Large Hadron Collider (LHC). The relentless search for new discoveries about the universe, demands an ever-increasing rate of collisions to increase the likelihood of capturing rare events. To achieve this goal, the LHC is in an upgrade phase to a high-luminosity Large Hadron Collider (HL-LHC). To address the challenges posed by the escalating collision rate, a new layer known as the Minimum Ionizing Particle Timing Detector (MTD) is incorporated into the CMS system. The MTD detector includes electronics positioned on the cylindrical barrel known as the barrel timing layer (BTL) and the end-cap timing layer (ETL) at the detector's ends. The MTD electronic system comprises approximately 66 thousand diagnostic sensors, including temperature, voltage, and current. These sensors help monitor the functionality of MTD data acquisition (MTD DAQ). The large number of sensors presents a major challenge for human operators who must analyze and interpret extensive sensor data. Hence, an important initial task of this work consisted of developing the MTD services software that offers a centralized system for fusing and compactly presenting the massive sensor data towards controlling and monitoring the MTD operations, keeping it safely operating close to its optimum point. This included the development of extensive code using Python and C++ to help in the control, monitoring, and graphical representation of this massive sensor data. Another important goal of this work was to maintain the integrity of the large sensor array by detecting anomalies and triggering corrective measures before possible problems escalations occur. This is particularly important in light of the very harsh operation environment the sensors are subjected to, namely extreme temperature and radiation. For this end, this work focused on four architectures of the autoencoder anomaly detection method, namely Convolutional Neural Network (CNN), 2D Convolutional with Long Short-Term Memory (Conv2DLSTM), CNN with U-net architecture, and Conv2DLSTM with U-net architecture. The latter method turned out to provide the best performance with an accuracy of 96.2% on simulated data. The choice of autoencoders was justified by their inherent denoising capabilities. This fact was supported by computer simulations indicating that the proposed anomaly detection algorithm is quite robust against additive noise as its performance was found to degrade by only about 1% with a drop in SNR by 12 dB. As a further means of noise mitigation, the wavelet denoising algorithm was applied to the sensor data before feeding it to the algorithm. Its success was marginal especially at low SNR since it caused some blurring of the anomalies hindering their detection. Finally, another important target of this project was to gather real sensor data in order to validate the simulated results. However, this task was rather challenging, and as of the writing of this work and only partial data was obtained. Initial results of applying the algorithm to the partial real data in hand showed a comparable accuracy of 94.5%.

k-means One-class classification performance was inadequate due to the lack of compact clusters in anomalous and normal latent space vectors. Further research using publicly available datasets is recommended to determine its efficacy and suitability.

5.2 Future points of expansion

As per the current requirements, this work only performs a binary classification of the sensor signals as either "*Abnormal*" or "*Normal*." As the MTD DAQ is in a development phase, a more elaborate scheme of dealing with anomalies is needed, such as the ability to identify the faulty sensors. For example Regional Convolution neural networks (R-CNN) can be used for this purpose as this capability has been demonstrated in the visual quality control

inspection of silicon sensors [14]. Furthermore, as more cases of anomalous behaviors will be known through the MTD DAQ development phase, the anomaly detector can be expanded to prompt suggestions on how to resolve the issues. Moreover, including a forecasting feature within the system may be useful to shorten the response time as it was demonstrated in health monitoring of the Hadron Calorimeter [17].

Bibliography

- [1] *Technical proposal*, ser. LHC technical proposal. Geneva: CERN, 1994, cover title : CMS, the Compact Muon Solenoid : technical proposal. [Online]. Available: <https://cds.cern.ch/record/290969>
- [2] Mar 2024. [Online]. Available: <https://home.cern/about>
- [3] T. C. Collaboration, “The Phase-2 Upgrade of the CMS Beam Radiation, Instrumentation, and Luminosity Detectors: Conceptual Design,” CERN, Geneva, Tech. Rep., 2020. [Online]. Available: <http://cds.cern.ch/record/2706512>
- [4] April 2023. [Online]. Available: <http://lhc-commissioning.web.cern.ch/schedule/LHC-long-term.htm>
- [5] F. Santanastasio, “Precision Timing in the CMS MTD Barrel Timing Layer With Crystal Bars and SiPMs,” *IEEE Trans. Nucl. Sci.*, vol. 67, no. 9, pp. 2105–2110, 2020. [Online]. Available: <https://cds.cern.ch/record/2772039>
- [6] I. Dutta, “Precision Timing with the CMS MIP detector,” Tech. Rep., 2018, * Temporary entry *. [Online]. Available: <https://cds.cern.ch/record/2641651>
- [7] M. Tornago, “The CMS MTD Endcap Timing Layer: Precision Timing with Low Gain Avalanche Detectors,” CERN, Geneva, Tech. Rep., 2023. [Online]. Available: <http://cds.cern.ch/record/2841557>
- [8] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, “Deep learning sensor fusion for autonomous vehicle perception and localization: A review,” *Sensors*, vol. 20, no. 15, 2020. [Online]. Available: <https://www.mdpi.com/1424-8220/20/15/4220>
- [9] M. L. Fung, M. Z. Q. Chen, and Y. H. Chen, “Sensor fusion: A review of methods and applications,” in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 3853–3860.
- [10] S. Kanjanawattana, “A novel outlier detection applied to an adaptive k-means,” *International Journal of Machine Learning and Computing*, vol. 9, pp. 569–574, 2019.
- [11] Y. Xiao, H. Wang, W. Xu, and J. Zhou, “Robust one-class svm for fault detection,” *Chemometrics and Intelligent Laboratory Systems*, vol. 151, pp. 15–25, 2016.
- [12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.
- [13] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.
- [14] S. Grönroos, M. Pierini, and N. Chernyavskaya, “Automated visual inspection of cms hgc al silicon sensor surface using an ensemble of a deep convolutional autoencoder and classifier,” *Machine Learning: Science and Technology*, vol. 4, no. 3, p. 035028, aug 2023. [Online]. Available: <https://dx.doi.org/10.1088/2632-2153/aced7e>
- [15] N. Zipper, “Testing a Neural Network for Anomaly Detection in the CMS Level-1 Global Trigger test crate during Run 3,” CERN, Geneva, Tech. Rep., 2023, 6 pages, 5 figures, submitted to JINST for inclusion in TWEPP 2023 conference proceedings. [Online]. Available: <https://cds.cern.ch/record/2883618>
- [16] —, “Testing a neural network for anomaly detection in the cms global trigger test crate during run 3,” 2023.
- [17] M. W. Asres, “Long horizon anomaly prediction in multivariate time series with causal autoencoders,” *PHME_CONF*, vol. 7, no. 1, pp. 21–31, 2022.
- [18] A. Papanastassiou, “Autoencoders for per-lumi-section data quality monitoring of the CMS detector,” CERN, Geneva, Tech. Rep., 2024. [Online]. Available: <https://cds.cern.ch/record/2886424>
- [19] K. R. Deja, “Using Machine Learning techniques for Data Quality Monitoring in CMS and ALICE experiments,” *PoS*, vol. LHCP2019, p. 236, 2019. [Online]. Available: <http://cds.cern.ch/record/2707754>

- [20] Y. Liu, J. Kang, C. Guo, Y. Bai, and S. Meng, "Fault diagnosis algorithm of gearbox based on napso-vmd self-adaptive noise reduction and dual-sensor feature fusion," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–22, 2022.
- [21] Q. Ye, S. Liu, and C. Liu, "A deep learning model for fault diagnosis with a deep neural network and feature fusion on multi-channel sensory signals," *Sensors*, vol. 20, no. 15, 2020. [Online]. Available: <https://www.mdpi.com/1424-8220/20/15/4300>
- [22] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, p. 78–96, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1566253517307844>
- [23] K. Zhang, Y. Li, W. Zuo, L. Zhang, L. V. Gool, and R. Timofte, "Plug-and-play image restoration with deep denoiser prior," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 6360–6376, 2022.
- [24] J. Yu, Y. Xu, and K. Liu, "Planetary gear fault diagnosis using stacked denoising autoencoder and gated recurrent unit neural network under noisy environment and time-varying rotational speed conditions," *Measurement Science and Technology*, vol. 30, p. 095003, 2019.
- [25] E. Hosseini-Asl, R. Keynton, and A. El-Baz, "Alzheimer's disease diagnostics by adaptation of 3d convolutional network," *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.
- [26] P. Goswami, M. Prakash, R. K. Ranjan, and A. Prakash, "A hybrid deep learning model for multi-step ahead prediction of pm2. 5 concentration across india," *Environmental Modeling & Assessment*, vol. 28, no. 5, pp. 803–816, 2023.
- [27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science*, pp. 234–241, 2015.
- [28] J. G. Munyua, G. M. Wambugu, and S. Njenga, "A survey of deep learning solutions for anomaly detection in surveillance videos," *International Journal of Computer and Information Technology(2279-0764)*, vol. 10, 2021.
- [29] G. Chen, W. Xie, and Y. Zhao, "Wavelet-based denoising: A brief review," in *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, 2013, pp. 570–574.
- [30] Feb 2024. [Online]. Available: <https://home.cern/news/press-release/cern/first-beam-lhc-accelerating-science>
- [31] L. R. Evans, "The large hadron collider," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, pp. 831–858, 2012.
- [32] O. Kodolova and M. Murray, "Heavy Ion Physics with CMS. Heavy Ion Physics Program in CMS Experiment," CERN, Geneva, Tech. Rep., 2009, comments: 8 pages, 8 figures - To appear in the conference proceedings for Quark Matter 2009, March 30 - April 4, Knoxville, Tennessee. [Online]. Available: <https://cds.cern.ch/record/1204545>
- [33] C. Collaboration, "The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger," CERN, Geneva, Tech. Rep., 2021, this is the final version of the document, approved by the LHCC. [Online]. Available: <http://cds.cern.ch/record/2759072>
- [34] C. Collaboration and A. Holzner, "78 reconstructed vertices in event from high-pileup run 198609," 2012, cMS Collection. [Online]. Available: <https://cds.cern.ch/record/1479324>
- [35] M. Malberti, "Precision timing with lyso:ce crystals and sipm sensors in the cms mtd barrel timing layer," *Journal of Instrumentation*, vol. 15, pp. C04 014–C04 014, 2020.
- [36] S. Collaboration, "EMP framework guide — EMP Framework Guide FW 0.8.2 / SW 0.8.2 documentation," 2020. [Online]. Available: <https://serenity.web.cern.ch/serenity/emp-fwk/>
- [37] MTD CMS Meeting - speaker: Mehmet Ozgur Sahin, "Backend system validation (hgcal workshop)," [Online] (CERN login required), 2023, [Accessed 13-04-2024]. [Online]. Available: <https://indico.cern.ch/event/1240449/>
- [38] MTD CMS Meeting- speaker: Hasan aljamea, "MTD XDAQ update," [Online] (CERN login required), April 2023, [Accessed 13-03-2024]. [Online]. Available: <https://indico.cern.ch/event/1274300/#10-mtd-xdaq-update>
- [39] S. Ramírez, "Fastapi," <https://fastapi.tiangolo.com/>, 2019.
- [40] A. Almarzouqi, "MTD Data Acquisition Graphical User Interface," 2023. [Online]. Available: <https://cds.cern.ch/record/2869906>
- [41] E. You and V. contributors, "Vue.js - the progressive javascript framework," 2023, available at <https://vuejs.org> [Online]. Available: <https://vuejs.org>
- [42] W. Xie, S. Uhlmann, S. Kiranyaz, and M. Gabbouj, "Incremental learning with support vector data description," *2014 22nd International Conference on Pattern Recognition*, 2014.

- [43] W. Ge and M. Sun, “An incremental two-dimensional principal component analysis for object recognition,” *Mathematical Problems in Engineering*, vol. 2018, pp. 1–13, 2018.
- [44] V. R. Kompella, M. Luciw, and J. Schmidhuber, “Incremental slow feature analysis: adaptive low-complexity slow feature updating from high-dimensional input streams,” *Neural Computation*, vol. 24, pp. 2994–3024, 2012.
- [45] G. M. Oxberry, T. Kostova-Vassilevska, W. Arrighi, and K. Chand, “Limited-memory adaptive snapshot selection for proper orthogonal decomposition,” *International Journal for Numerical Methods in Engineering*, vol. 109, pp. 198–217, 2016.
- [46] A. F. Gad, “Pygad: An intuitive genetic algorithm python library,” *Multimedia Tools and Applications*, pp. 1–14, 2023.
- [47] G. R. Lee, R. Gommers, F. Waselewski, K. J. Wohlfahrt, and A. O’Leary, “Pywavelets: a python package for wavelet analysis,” *Journal of Open Source Software*, vol. 4, p. 1237, 2019.
- [48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [50] R. Kress, *The Heat Equation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, p. 132–141. [Online]. Available: https://doi.org/10.1007/978-3-642-97146-4_9
- [51] A. Somer, F. Camilotti, G. F. Costa, C. Bonardi, A. Novatski, A. V. C. Andrade, J. Kozlowski, V. A., and G. K. Cruz, “The thermoelastic bending and thermal diffusion processes influence on photoacoustic signal generation using open photoacoustic cell technique,” *Journal of Applied Physics*, vol. 114, no. 6, p. 063503, 08 2013. [Online]. Available: <https://doi.org/10.1063/1.4817655>
- [52] B. Marzocchi, “Cms mtd barrel timing layer: precision timing at the hl-lhc,” *Proceedings of the European Physical Society Conference on High Energy Physics — PoS(EPS-HEP2021)*, 2022.

Appendix A: Formal Mathematical Derivations

A.1 Discrete form of the Heat equation

This section will drive the Discrete form of the Heat equation to be implemented to iteratively calculate the next time frame of the Temperature sensor window based on the values in the current time frame. The heat partial differential equation (PDE) for 2D spatial surfaces with PDE notation:

$$u_t = \alpha(u_{xx} + u_{yy}) \quad (\text{A.1})$$

And more explicitly:

$$\frac{\partial u(t, x, y)}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \quad (\text{A.2})$$

where

$u(t, x, y)$: The heat as a function of time “t” and the spatial dimensions “x” and “y”.

α : The thermal diffusivity in unit of $cm^2 s^{-1}$.

Let $u_{x,y}^n$ denote the discrete version of $u(t, x, y)$ where “n” denoting the discrete time index. Differential approximations will be used to approximate value of $u_{x,y}^{n+1}$ base on values from “n” time frame.

For $f(x)$

$$\begin{aligned} \frac{df(x)}{dx} &= \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \\ &\approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \end{aligned} \quad \text{let } h = \Delta x \quad (\text{A.3})$$

$$\begin{aligned} \frac{d^2 f(x)}{dx^2} &= \frac{\left(\frac{df(x)}{dx} \right)}{dx} \\ &= \lim_{h \rightarrow 0} \frac{\left(\frac{f(x+2h) - f(x+h)}{h} \right) - \left(\frac{f(x+h) - f(x)}{h} \right)}{h} \\ &= \lim_{h \rightarrow 0} \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \\ &\approx \frac{f(x+2\Delta x) - 2f(x+\Delta x) + f(x)}{\Delta x^2} \end{aligned} \quad \text{let } h = \Delta x \quad (\text{A.4})$$

Using differential approximations [A.3](#) and [A.4](#) on the partial differentials in equation [A.2](#) starting with $\frac{\partial u(t, x, y)}{\partial t}$ term:

$$\frac{\partial u(t, x, y)}{\partial t} \approx \frac{u(t + \Delta t, x, y) - u(t, x, y)}{\Delta t} \quad (\text{A.5})$$

Since time difference between time frame “n” and “n + 1” is Δt :

$$\frac{\partial u(t, x, y)}{\partial t} \approx \frac{u_{x,y}^{n+1} - u_{x,y}^n}{\Delta t} \quad (\text{A.6})$$

Now approximating the term $\frac{\partial^2 u(t,x,y)}{\partial x^2}$:

$$\frac{\partial^2 u(t,x,y)}{\partial x^2} \approx \frac{u(t,x+2\Delta x,y) - 2u(t,x+\Delta x,y) + u(t,x,y)}{(\Delta x)^2} \quad (\text{A.7})$$

Considering the distance between spatial frame “ x ” and “ $x+1$ ” is Δx :

$$\frac{\partial^2 u(t,x,y)}{\partial x^2} \approx \frac{u_{x+2,y}^n - 2u_{x+1,y}^n + u_{x,y}^n}{(\Delta x)^2} \quad (\text{A.8})$$

Similarly with the term $\frac{\partial^2 u(t,x,y)}{\partial y^2}$:

$$\frac{\partial^2 u(t,x,y)}{\partial y^2} \approx \frac{u_{x,y+2}^n - 2u_{x,y+1}^n + u_{x,y}^n}{(\Delta y)^2} \quad (\text{A.9})$$

Substitute equations [A.7](#)[A.8](#)[A.9](#) into equation [A.2](#)

$$\frac{u_{x,y}^{n+1} - u_{x,y}^n}{\Delta t} \approx \alpha \left(\frac{u_{x+2,y}^n - 2u_{x+1,y}^n + u_{x,y}^n}{(\Delta x)^2} + \frac{u_{x,y+2}^n - 2u_{x,y+1}^n + u_{x,y}^n}{(\Delta y)^2} \right) \quad (\text{A.10})$$

$$u_{x,y}^{n+1} \approx u_{x,y}^n + (\Delta t)\alpha \left(\frac{u_{x+2,y}^n - 2u_{x+1,y}^n + u_{x,y}^n}{(\Delta x)^2} + \frac{u_{x,y+2}^n - 2u_{x,y+1}^n + u_{x,y}^n}{(\Delta y)^2} \right) \quad (\text{A.11})$$

A.2 Updating variance σ^2 and mean μ of scalar data

This formulation aims at updating the variance and mean in a substantially large scalar dataset; hence, it's done in batches.

Let the current dataset of size “ n ” $\{x_i, i = 1 \dots n\}$ be compactly denoted by $\{X\}_{1:n}$ and the new dataset of size “ m ” $\{y_i, i = 1 \dots m\}$ be compactly denoted by $\{Y\}_{1:m}$. The individual dataset means and variances are given by:

$$\mu_x = \frac{1}{n} \sum_{i=0}^n X_i \quad \mu_y = \frac{1}{m} \sum_{i=0}^m Y_i \quad (\text{A.12})$$

$$\sigma_x^2 = \frac{1}{n-1} \sum_{i=0}^n (X_i - \mu_x)^2 \quad \sigma_y^2 = \frac{1}{m-1} \sum_{i=0}^m (Y_i - \mu_y)^2 \quad (\text{A.13})$$

The objective is to update the data statistics by calculating the mean μ and variance σ^2 for the augmented dataset, which is the concatenation of both datasets

$$\{Z\}_{1:(n+m)} = \{X\}_{1:n} \cup \{Y\}_{1:m}$$

However, only $\{\mu_x, \sigma_x^2, \mu_y, \sigma_y^2, \{Y\}_{1:m}\}$ are known where the datasets $\{X\}_{1:n}$ is discarded to save memory.

Updating the mean μ is as follows:-

$$\begin{aligned} \mu &= \frac{\sum_{k=1}^{(n+m)} Z_k}{n+m} \\ &= \frac{\sum_{i=1}^n X_i + \sum_{j=1}^m Y_j}{n+m} \\ &= \frac{n\mu_x + m\mu_y}{m+n} \end{aligned} \quad (\text{A.14})$$

Updating the σ^2 is as follows:

$$\sigma^2 = \frac{1}{n+m-1} \left(\sum_{i=1}^n (X_i - \mu)^2 + \sum_{j=1}^m (Y_j - \mu)^2 \right) \quad (\text{A.15})$$

$$\begin{aligned} &= \frac{1}{n+m-1} \left(\sum_{i=0}^n (X_i - \mu_x + \mu_x - \mu)^2 + \sum_{j=0}^m (Y_j - \mu_y + \mu_y - \mu)^2 \right) \\ &= \frac{1}{n+m-1} \left(\sum_{i=0}^n (X_i - \mu_x)^2 + \sum_{i=0}^n 2(X_i - \mu_x)(\mu_x - \mu) + \sum_{i=0}^n (\mu_x - \mu)^2 \right. \\ &\quad \left. + \sum_{j=0}^m (Y_j - \mu_y)^2 + \sum_{j=0}^m 2(Y_j - \mu_y)(\mu_y - \mu) + \sum_{j=0}^m (\mu_y - \mu)^2 \right) \\ &= \frac{1}{n+m-1} \left(\begin{aligned} &= (n-1)\sigma_x^2 + 2(\cancel{n\mu_x} \xrightarrow{0} \cancel{n\mu_x})(\mu_x - \mu) + n(\mu_x - \mu)^2 \\ &+ (m-1)\sigma_y^2 + 2(\cancel{m\mu_y} \xrightarrow{0} \cancel{m\mu_y})(\mu_y - \mu) + m(\mu_y - \mu)^2 \end{aligned} \right) \\ &= \frac{(n-1)\sigma_x^2 + (m-1)\sigma_y^2 + n(\mu_x - \mu)^2 + m(\mu_y - \mu)^2}{n+m-1} \end{aligned} \quad (\text{A.16})$$

Simplifying the term $n(\mu_x - \mu)^2 + m(\mu_y - \mu)^2$:

$$\begin{aligned}
n(\mu_x - \mu)^2 + m(\mu_y - \mu)^2 &= n \left(\mu_x - \frac{n\mu_x + m\mu_y}{m+n} \right)^2 + m \left(\mu_y - \frac{n\mu_x + m\mu_y}{m+n} \right)^2 \\
&= n \left(\frac{\mu_x(n+m) - n\mu_x - m\mu_y}{m+n} \right)^2 + m \left(\frac{\mu_y(n+m) - n\mu_x - m\mu_y}{m+n} \right)^2 \\
&= \frac{1}{(m+n)^2} \left(n(m\mu_x + \cancel{n\mu_x}^0 - m\mu_y)^2 + m(n\mu_y + \cancel{m\mu_y}^0 - n\mu_x)^2 \right) \\
&= \frac{1}{(m+n)^2} (n(m\mu_x - m\mu_y)^2 + m(n\mu_y - n\mu_x)^2) \\
&= \frac{1}{(m+n)^2} (nm^2(\mu_x - \mu_y)^2 + n^2m(\mu_y - \mu_x)^2) \\
&= \frac{nm}{(m+n)^2} (m(\mu_x - \mu_y)^2 + n(\mu_y - \mu_x)^2) \\
&= \frac{nm(m+n)(\mu_x - \mu_y)^2}{(m+n)^2} \\
&= \frac{nm(\mu_x - \mu_y)^2}{m+n}
\end{aligned} \tag{A.17}$$

Finally, The variance of the augmented dataset is:

$$\sigma^2 = \frac{(n-1)\sigma_x^2 + (m-1)\sigma_y^2 + \frac{mn}{n+m}(\mu_x - \mu_y)^2}{n+m-1} \tag{A.18}$$

A.3 Updating Covariance matrix and mean of vector data

The following derivations are synonymous to the previous ones, except they apply for the vector dataset statistics, namely the covariance matrix and vector mean. Let the current vector dataset (each vector is of dimension “ k ”) of size “ n ” $\{\vec{x}_i, i = 1 \dots n\}$ compactly denoted by $\{\vec{X}\}_{1:n}$ and the new vector dataset (each vector is of dimension “ k ”) of size “ m ” $\{\vec{y}_i, i = 1 \dots m\}$ compactly denoted by $\{\vec{Y}\}_{1:m}$.

By ordering the data vectors as rows of matrices, we utilize the convenience of matrix notations as follows

$$\mathbf{X} = \left[\{\vec{X}\}_{1:n} \right]^T, n \times k \text{ matrix} \tag{A.19}$$

$$\mathbf{Y} = \left[\{\vec{Y}\}_{1:m} \right]^T, m \times k \text{ matrix} \tag{A.20}$$

The row vector means of both the current and new data matrices are obtained by averaging the columns of $\mathbf{X}$ and $\mathbf{Y}$.

$$\begin{aligned}
\vec{\mu}_x &= \frac{1}{n}(\mathbf{1}_{1 \times n} \mathbf{X}), & 1 \times k \text{ matrix} \\
\vec{\mu}_y &= \frac{1}{m}(\mathbf{1}_{1 \times m} \mathbf{Y}), & 1 \times k \text{ matrix}
\end{aligned} \tag{A.21}$$

where $\mathbf{1}_{1 \times n}$ and $\mathbf{1}_{1 \times m}$ denote the all-one row vectors with dimensions n and m , respectively. The mean matrices are obtained concatenating the above mean vectors as follows:

$$\begin{aligned}
\bar{\mathbf{X}} &= \mathbf{1}_{n \times 1} \vec{\mu}_x, & n \times k \text{ matrix} \\
\bar{\mathbf{Y}} &= \mathbf{1}_{m \times 1} \vec{\mu}_y, & m \times k \text{ matrix}
\end{aligned} \tag{A.22}$$

where $\mathbf{1}_{n \times 1}$ and $\mathbf{1}_{m \times 1}$ denote the all-one column vectors with dimensions n and m , respectively.

We further define

$$\begin{aligned}
\hat{\mathbf{X}} &= \mathbf{X} - \bar{\mathbf{X}}, & n \times k \text{ matrix} \\
\hat{\mathbf{Y}} &= \mathbf{Y} - \bar{\mathbf{Y}}, & m \times k \text{ matrix}
\end{aligned} \tag{A.23}$$

Hence, the Covariance matrices for both the current and new datasets may be expressed as:

$$\begin{aligned} \mathbf{C}_x &= \frac{1}{n-1} \hat{\mathbf{X}}^T \hat{\mathbf{X}}, k \times k \text{ matrix} \\ \mathbf{C}_y &= \frac{1}{m-1} \hat{\mathbf{Y}}^T \hat{\mathbf{Y}}, k \times k \text{ matrix} \end{aligned} \quad (\text{A.24})$$

Let the augmented vector dataset be denoted by $\{\vec{Z}\}_{1:(n+m)} = \{\vec{X}\}_{1:n} \cup \{\vec{Y}\}_{1:m}$. We further augment the data matrices $\mathbf{X}$ and $\mathbf{Y}$ to obtain the matrix $\mathbf{Z}$ as

$$\mathbf{Z} = \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix}, (n+m) \times k \text{ matrix} \quad (\text{A.25})$$

The augmented data mean vector $\vec{\mu}_z$ is given as in Eq [A.21](#) as:

$$\vec{\mu}_z = \frac{1}{n} (\mathbf{1}_{1 \times n} \mathbf{Z}), 1 \times k \text{ vector} \quad (\text{A.26})$$

And is related to $\vec{\mu}_x$ and $\vec{\mu}_y$ by

$$\vec{\mu}_z = \frac{n\vec{\mu}_x + m\vec{\mu}_y}{n+m}, 1 \times k \text{ vector} \quad (\text{A.27})$$

With its mean matrix given by

$$\begin{aligned} \bar{\mathbf{Z}} &= \mathbf{1}_{(n+m) \times 1} \vec{\mu}_z, (n+m) \times k \text{ matrix} \\ &= \begin{bmatrix} \mathbf{1}_{n \times 1} \vec{\mu}_z \\ \mathbf{1}_{m \times 1} \vec{\mu}_z \end{bmatrix} \end{aligned} \quad (\text{A.28})$$

As in eq [A.23](#) we define matrix $\hat{\mathbf{Z}}$ as

$$\begin{aligned} \hat{\mathbf{Z}} &= \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} - \bar{\mathbf{Z}} \\ &= \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} - \begin{bmatrix} \bar{\mathbf{X}} \\ \bar{\mathbf{Y}} \end{bmatrix} + \begin{bmatrix} \bar{\mathbf{X}} \\ \bar{\mathbf{Y}} \end{bmatrix} - \bar{\mathbf{Z}} \\ &= \begin{bmatrix} \hat{\mathbf{X}} \\ \hat{\mathbf{Y}} \end{bmatrix} + \begin{bmatrix} \mathbf{1}_{n \times 1} \vec{\mu}_x \\ \mathbf{1}_{m \times 1} \vec{\mu}_y \end{bmatrix} - \bar{\mathbf{Z}} \\ &= \begin{bmatrix} \hat{\mathbf{X}} \\ \hat{\mathbf{Y}} \end{bmatrix} + \begin{bmatrix} \mathbf{1}_{n \times 1} (\vec{\mu}_x - \vec{\mu}_z) \\ \mathbf{1}_{m \times 1} (\vec{\mu}_y - \vec{\mu}_z) \end{bmatrix} \end{aligned} \quad (\text{A.29})$$

The covariance matrix:

$$\mathbf{C}_z = \frac{1}{n+m-1} \hat{\mathbf{Z}}^T \hat{\mathbf{Z}} \quad (\text{A.30})$$

$$\begin{aligned} &= \frac{1}{n+m-1} \left(\begin{bmatrix} \hat{\mathbf{X}}^T & \hat{\mathbf{Y}}^T \end{bmatrix} + \begin{bmatrix} (\vec{\mu}_x - \vec{\mu}_z)^T \mathbf{1}_{(1 \times n)} & (\vec{\mu}_y - \vec{\mu}_z)^T \mathbf{1}_{(1 \times m)} \end{bmatrix} \right) \\ &\quad \left(\begin{bmatrix} \hat{\mathbf{X}} \\ \hat{\mathbf{Y}} \end{bmatrix} + \begin{bmatrix} \mathbf{1}_{(n \times 1)} (\vec{\mu}_x - \vec{\mu}_z) \\ \mathbf{1}_{(m \times 1)} (\vec{\mu}_y - \vec{\mu}_z) \end{bmatrix} \right) \end{aligned} \quad (\text{A.31})$$

$$\begin{aligned} &= \frac{1}{n+m-1} \left(\begin{aligned} &\hat{\mathbf{X}}^T \hat{\mathbf{X}} + \hat{\mathbf{Y}}^T \hat{\mathbf{Y}} \\ &+ \begin{bmatrix} \hat{\mathbf{X}}^T \mathbf{1}_{n \times 1} (\vec{\mu}_x - \vec{\mu}_z) \\ \hat{\mathbf{Y}}^T \mathbf{1}_{m \times 1} (\vec{\mu}_y - \vec{\mu}_z) \end{bmatrix} + \begin{bmatrix} (\vec{\mu}_x - \vec{\mu}_z)^T \mathbf{1}_{(n \times 1)} \hat{\mathbf{X}}^T \\ (\vec{\mu}_y - \vec{\mu}_z)^T \mathbf{1}_{(m \times 1)} \hat{\mathbf{Y}}^T \end{bmatrix} \\ &+ (\vec{\mu}_x - \vec{\mu}_z)^T (\vec{\mu}_x - \vec{\mu}_z) + (\vec{\mu}_y - \vec{\mu}_z)^T (\vec{\mu}_y - \vec{\mu}_z) \end{aligned} \right) \end{aligned} \quad (\text{A.32})$$

$$\begin{aligned} &+ \begin{bmatrix} \hat{\mathbf{X}}^T \mathbf{1}_{n \times 1} (\vec{\mu}_x - \vec{\mu}_z) \\ \hat{\mathbf{Y}}^T \mathbf{1}_{m \times 1} (\vec{\mu}_y - \vec{\mu}_z) \end{bmatrix} + \begin{bmatrix} (\vec{\mu}_x - \vec{\mu}_z)^T \mathbf{1}_{(n \times 1)} \hat{\mathbf{X}}^T \\ (\vec{\mu}_y - \vec{\mu}_z)^T \mathbf{1}_{(m \times 1)} \hat{\mathbf{Y}}^T \end{bmatrix} \\ &+ (\vec{\mu}_x - \vec{\mu}_z)^T (\vec{\mu}_x - \vec{\mu}_z) + (\vec{\mu}_y - \vec{\mu}_z)^T (\vec{\mu}_y - \vec{\mu}_z) \end{aligned} \quad (\text{A.33})$$

The Middle term is zero because:

$$\begin{aligned}
\mathbf{1}_{1 \times n} \hat{\mathbf{X}} &= \mathbf{1}_{1 \times n} (\mathbf{X} - \bar{\mathbf{X}}) \\
&= \mathbf{1}_{1 \times n} \mathbf{X} - \mathbf{1}_{1 \times n} \bar{\mathbf{X}} \\
&= n\vec{\mu}_x - \mathbf{1}_{1 \times n} \mathbf{1}_{n \times 1} \vec{\mu}_x \\
&= n\vec{\mu}_x - n\vec{\mu}_x \\
&= \vec{\mathbf{0}}_{1 \times k}
\end{aligned} \tag{A.34}$$

$$\begin{aligned}
\hat{\mathbf{X}}^T \mathbf{1}_{n \times 1} &= (\mathbf{1}_{1 \times n} \hat{\mathbf{X}})^T \\
&= \vec{\mathbf{0}}_{k \times 1}
\end{aligned} \tag{A.35}$$

Similarly

$$\begin{aligned}
\hat{\mathbf{Y}}^T \mathbf{1}_{m \times 1} &= \vec{\mathbf{0}}_{k \times 1} \\
\mathbf{1}_{m \times 1} \hat{\mathbf{Y}} &= \vec{\mathbf{0}}_{1 \times k}
\end{aligned} \tag{A.36}$$

Where $\vec{\mathbf{0}}_{1 \times k}$ and $\vec{\mathbf{0}}_{k \times 1}$ are zero row and column vectors.

$$\begin{aligned}
\mathbf{C}_z &= \frac{1}{n+m-1} \left(\hat{\mathbf{X}}^T \hat{\mathbf{X}} + \hat{\mathbf{Y}}^T \hat{\mathbf{Y}} + (\vec{\mu}_x - \vec{\mu}_z)^T (\vec{\mu}_x - \vec{\mu}_z) + (\vec{\mu}_y - \vec{\mu}_z)^T (\vec{\mu}_y - \vec{\mu}_z) \right) \\
&= \frac{1}{n+m-1} \left((n-1)\mathbf{C}_x + (m-1)\mathbf{C}_y + (\vec{\mu}_x - \vec{\mu}_z)^T (\vec{\mu}_x - \vec{\mu}_z) + (\vec{\mu}_y - \vec{\mu}_z)^T (\vec{\mu}_y - \vec{\mu}_z) \right)
\end{aligned} \tag{A.37}$$

Simplifying terms $(\vec{\mu}_x - \vec{\mu}_z)$ and $(\vec{\mu}_y - \vec{\mu}_z)$:

$$\begin{aligned}
\vec{\mu}_x - \vec{\mu}_z &= \vec{\mu}_x - \frac{n\vec{\mu}_x + m\vec{\mu}_y}{n+m} \\
&= \frac{\vec{\mu}_x(n+m) - n\vec{\mu}_x - m\vec{\mu}_y}{n+m} \\
&= \frac{n\vec{\mu}_x + m\vec{\mu}_x - n\vec{\mu}_x - m\vec{\mu}_y}{n+m} \\
&= \frac{m}{n+m} (\vec{\mu}_x - \vec{\mu}_y)
\end{aligned} \tag{A.38}$$

$$\begin{aligned}
\vec{\mu}_y - \vec{\mu}_z &= \vec{\mu}_y - \frac{n\vec{\mu}_x + m\vec{\mu}_y}{n+m} \\
&= \frac{\vec{\mu}_y(n+m) - n\vec{\mu}_x - m\vec{\mu}_y}{n+m} \\
&= \frac{n\vec{\mu}_y + m\vec{\mu}_y - n\vec{\mu}_x - m\vec{\mu}_y}{n+m} \\
&= \frac{-n}{n+m} (\vec{\mu}_x - \vec{\mu}_y)
\end{aligned} \tag{A.39}$$

$$\begin{aligned}
\mathbf{C}_z &= \frac{1}{n+m-1} \left((n-1)\mathbf{C}_x + (m-1)\mathbf{C}_y \right. \\
&\quad \left. + \left(\frac{m}{n+m}\right)^2 (\vec{\mu}_x - \vec{\mu}_y)^T (\vec{\mu}_x - \vec{\mu}_y) + \left(\frac{n}{n+m}\right)^2 (\vec{\mu}_x - \vec{\mu}_y)^T (\vec{\mu}_x - \vec{\mu}_y) \right)
\end{aligned} \tag{A.40}$$

Finally, the Augmented dataset Covariance matrix:

$$\mathbf{C}_Z = \frac{1}{n+m-1} \left((n-1)\mathbf{C}_x + (m-1)\mathbf{C}_y + \frac{n^2 + m^2}{(n+m)^2} (\vec{\mu}_x - \vec{\mu}_y)^T (\vec{\mu}_x - \vec{\mu}_y) \right) \tag{A.41}$$

Appendix B: BTL Readout Unit full sensor list

The table [B.1](#) contains a comprehensive list of BTL Sensors, along with information about their placement, the source of their readings (the ASIC chip), and their range.

Table B.1: BTL Readout Unit full sensor list

Sensor on	Source	Sensor name	Range
CC Board	LPGBT L0	CC Board Temp 1	Pt1000
CC Board	LPGBT L1	CC Board Temp 2	Pt1000
CC Board	LPGBT L1	CC Board Temp 3	Pt1000
CC Board	GBT-SCA A	V Bias A	(0-61)v
CC Board	GBT-SCA B	V Bias B	(0-61)v
CC Board	LPGBT L0	VDDA A Voltage	(0-2.5)v
CC Board	LPGBT L0	VDDA B Voltage	(0-2.5)v
CC Board	LPGBT L1	VDDA C Voltage	(0-2.5)v
CC Board	LPGBT L1	VDDA D Voltage	(0-2.5)v
CC Board	LPGBT L0	Vin Voltage	(0-15)v
FE01 BTL Module	GBT-SCA A	FE1 Bias Current 1	(0-2.247) or (0-44.94) mA
FE01 BTL Module	GBT-SCA A	FE1 Bias Current 2	(0-2.247) or (0-44.94) mA
FE01 BTL Module	GBT-SCA A	FE1 Bias Current 3	(0-2.247) or (0-44.94) mA
FE01 BTL Module	GBT-SCA A	FE1 Bias Current 4	(0-2.247) or (0-44.94) mA
FE01 BTL Module	GBT-SCA A	FE1 SiPM Temp 1	Pt1000
FE01 BTL Module	GBT-SCA A	FE1 SiPM Temp 2	Pt1000
FE02 BTL Module	GBT-SCA A	FE2 Bias Current 1	(0-2.247) or (0-44.94) mA
FE02 BTL Module	GBT-SCA A	FE2 Bias Current 2	(0-2.247) or (0-44.94) mA
FE02 BTL Module	GBT-SCA A	FE2 Bias Current 3	(0-2.247) or (0-44.94) mA
FE02 BTL Module	GBT-SCA A	FE2 Bias Current 4	(0-2.247) or (0-44.94) mA
FE02 BTL Module	GBT-SCA A	FE2 SiPM Temp 1	Pt1000
FE02 BTL Module	GBT-SCA A	FE2 SiPM Temp 2	Pt1000
FE03 BTL Module	GBT-SCA A	FE3 Bias Current 1	(0-2.247) or (0-44.94) mA
FE03 BTL Module	GBT-SCA A	FE3 Bias Current 2	(0-2.247) or (0-44.94) mA
FE03 BTL Module	GBT-SCA A	FE3 Bias Current 3	(0-2.247) or (0-44.94) mA
FE03 BTL Module	GBT-SCA A	FE3 Bias Current 4	(0-2.247) or (0-44.94) mA
FE04 BTL Module	GBT-SCA A	FE4 Bias Current 1	(0-2.247) or (0-44.94) mA
FE04 BTL Module	GBT-SCA A	FE4 Bias Current 2	(0-2.247) or (0-44.94) mA
FE04 BTL Module	GBT-SCA A	FE4 Bias Current 3	(0-2.247) or (0-44.94) mA
FE04 BTL Module	GBT-SCA A	FE4 Bias Current 4	(0-2.247) or (0-44.94) mA
FE05 BTL Module	GBT-SCA A	FE5 Bias Current 1	(0-2.247) or (0-44.94) mA
FE05 BTL Module	GBT-SCA A	FE5 Bias Current 2	(0-2.247) or (0-44.94) mA
FE05 BTL Module	GBT-SCA A	FE5 Bias Current 3	(0-2.247) or (0-44.94) mA
FE05 BTL Module	GBT-SCA A	FE5 Bias Current 4	(0-2.247) or (0-44.94) mA
FE05 BTL Module	GBT-SCA A	FE5 SiPM Temp 1	Pt1000
FE05 BTL Module	GBT-SCA A	FE5 SiPM Temp 2	Pt1000
FE06 BTL Module	GBT-SCA A	FE6 Bias Current 1	(0-2.247) or (0-44.94) mA
FE06 BTL Module	GBT-SCA A	FE6 Bias Current 2	(0-2.247) or (0-44.94) mA
FE06 BTL Module	GBT-SCA A	FE6 Bias Current 3	(0-2.247) or (0-44.94) mA

FE06 BTL Module	GBT-SCA A	FE6 Bias Current 4	(0-2.247) or (0-44.94) mA
FE06 BTL Module	LPGBT L0	FE6 SiPM Temp 1	Pt1000
FE06 BTL Module	LPGBT L0	FE6 SiPM Temp 2	Pt1000
FE07 BTL Module	GBT-SCA B	FE7 Bias Current 1	(0-2.247) or (0-44.94) mA
FE07 BTL Module	GBT-SCA B	FE7 Bias Current 2	(0-2.247) or (0-44.94) mA
FE07 BTL Module	GBT-SCA B	FE7 Bias Current 3	(0-2.247) or (0-44.94) mA
FE07 BTL Module	GBT-SCA B	FE7 Bias Current 4	(0-2.247) or (0-44.94) mA
FE07 BTL Module	GBT-SCA B	FE7 SiPM Temp 1	Pt1000
FE07 BTL Module	GBT-SCA B	FE7 SiPM Temp 2	Pt1000
FE08 BTL Module	GBT-SCA B	FE8 Bias Current 1	(0-2.247) or (0-44.94) mA
FE08 BTL Module	GBT-SCA B	FE8 Bias Current 2	(0-2.247) or (0-44.94) mA
FE08 BTL Module	GBT-SCA B	FE8 Bias Current 3	(0-2.247) or (0-44.94) mA
FE08 BTL Module	GBT-SCA B	FE8 Bias Current 4	(0-2.247) or (0-44.94) mA
FE08 BTL Module	GBT-SCA B	FE8 SiPM Temp 1	Pt1000
FE08 BTL Module	GBT-SCA B	FE8 SiPM Temp 2	Pt1000
FE09 BTL Module	GBT-SCA B	FE9 Bias Current 1	(0-2.247) or (0-44.94) mA
FE09 BTL Module	GBT-SCA B	FE9 Bias Current 2	(0-2.247) or (0-44.94) mA
FE09 BTL Module	GBT-SCA B	FE9 Bias Current 3	(0-2.247) or (0-44.94) mA
FE09 BTL Module	GBT-SCA B	FE9 Bias Current 4	(0-2.247) or (0-44.94) mA
FE10 BTL Module	GBT-SCA B	FE10 Bias Current 1	(0-2.247) or (0-44.94) mA
FE10 BTL Module	GBT-SCA B	FE10 Bias Current 2	(0-2.247) or (0-44.94) mA
FE10 BTL Module	GBT-SCA B	FE10 Bias Current 3	(0-2.247) or (0-44.94) mA
FE10 BTL Module	GBT-SCA B	FE10 Bias Current 4	(0-2.247) or (0-44.94) mA
FE11 BTL Module	GBT-SCA B	FE11 Bias Current 1	(0-2.247) or (0-44.94) mA
FE11 BTL Module	GBT-SCA B	FE11 Bias Current 2	(0-2.247) or (0-44.94) mA
FE11 BTL Module	GBT-SCA B	FE11 Bias Current 3	(0-2.247) or (0-44.94) mA
FE11 BTL Module	GBT-SCA B	FE11 Bias Current 4	(0-2.247) or (0-44.94) mA
FE11 BTL Module	GBT-SCA B	FE11 SiPM Temp 1	Pt1000
FE11 BTL Module	GBT-SCA B	FE11 SiPM Temp 2	Pt1000
FE12 BTL Module	GBT-SCA B	FE12 Bias Current 1	(0-2.247) or (0-44.94) mA
FE12 BTL Module	GBT-SCA B	FE12 Bias Current 2	(0-2.247) or (0-44.94) mA
FE12 BTL Module	GBT-SCA B	FE12 Bias Current 3	(0-2.247) or (0-44.94) mA
FE12 BTL Module	GBT-SCA B	FE12 Bias Current 4	(0-2.247) or (0-44.94) mA
FE12 BTL Module	LPGBT L1	FE12 SiPM Temp 1	Pt1000
FE12 BTL Module	LPGBT L1	FE12 SiPM Temp 2	Pt1000
PCC A	LPGBT L0	PCC A Temp 1	Pt1000
PCC A	LPGBT L0	PCC A Temp 2	Pt1000
PCC B	LPGBT L1	PCC B Temp 1	Pt1000
PCC B	LPGBT L1	PCC B Temp 2	Pt1000

Building the AutoEncoders using Tensorflow

The following Python code constructs the AutoEncoder architecture (CNN-AE, LSTM-AE, U-net CNN, U-net LSTM) and saves it as a JSON file for easy loading and training.

```
1 from typing import Tuple
2 from pathlib import Path
3 import tensorflow as tf
4 from keras import Model, Input, Sequential, layers
5 import json
6 import matplotlib.pyplot as plt
7
8
9 def skip_connection(layer: Model, input_shape: tuple = None):
10     if input_shape is None:
11         x_in = Input(shape=layer.input_shape[1:])
12     else:
13         x_in = Input(shape=input_shape)
14     x_out = layer(x_in)
15     x_out = tf.keras.layers.Add()([x_in, x_out])
16     return Model(inputs=x_in, outputs=x_out)
17
18
19 def ConvReshape(input_shape: Tuple[int, int, int, int],
20                 output_shape: Tuple[int, int, int, int],
21                 lstm: bool):
22     a, b, c, d = input_shape
23     aa, bb, cc, dd = output_shape
24     if lstm:
25         return Sequential([
26             layers.InputLayer((a, b, c, d)),
27             layers.Permute((1, 4, 2, 3)),
28             layers.ConvLSTM2D(
29                 filters=dd,
30                 kernel_size=3,
31                 strides=1,
32                 return_sequences=True,
33                 padding='same',
34                 activation='leaky_relu'
35             ),
36             layers.Permute((1, 4, 2, 3)),
37             layers.ConvLSTM2D(
38                 filters=cc,
39                 kernel_size=3,
40                 strides=1,
41                 return_sequences=True,
42                 padding='same',
43                 activation='leaky_relu'
44             ),
45             layers.Permute((1, 4, 2, 3)),
46             layers.ConvLSTM2D(
47                 filters=bb,
48                 kernel_size=3,
49                 strides=1,
50                 return_sequences=True,
51                 padding='same',
```

```

52         activation='leaky_relu'
53     ),
54     layers.Permute((1, 4, 2, 3)),
55 ]
56 else:
57     return Sequential([
58         layers.InputLayer((a, b, c, d)),
59         layers.Permute((1, 4, 2, 3)),
60         layers.Conv3D(
61             filters=dd,
62             kernel_size=3,
63             strides=1,
64             padding='same',
65             activation='leaky_relu'
66         ),
67         layers.Permute((1, 4, 2, 3)),
68         layers.Conv3D(
69             filters=cc,
70             kernel_size=3,
71             strides=1,
72             padding='same',
73             activation='leaky_relu'
74         ),
75         layers.Permute((1, 4, 2, 3)),
76         layers.Conv3D(
77             filters=bb,
78             kernel_size=3,
79             strides=1,
80             padding='same',
81             activation='leaky_relu'
82         ),
83         layers.Permute((1, 4, 2, 3)),
84     ])
85
86
87 def autoencoder(input_shape: Tuple[int, int, int, int], lstm: bool, name_prefix: str = '') -
88     > Model:
89     encoder = Sequential([
90         layers.InputLayer(input_shape=input_shape),
91         ConvReshape(input_shape, (input_shape[0], 16, 16, 64), lstm),
92     ])
93     if lstm:
94         encoder.add(skip_connection(layers.ConvLSTM2D(
95             filters=64,
96             kernel_size=(3, 3),
97             strides=(1, 1),
98             padding='same',
99             return_sequences=True,
100             activation='leaky_relu',
101             kernel_initializer=tf.random_normal_initializer(0., 0.02),
102         ),
103             (30, 16, 16, 64)
104         )),
105         encoder.add(layers.ConvLSTM2D(
106             filters=64,
107             kernel_size=(3, 3),
108             strides=(2, 2),
109             padding='same',
110             return_sequences=True,
111             activation='leaky_relu',
112             kernel_initializer=tf.random_normal_initializer(0., 0.02),
113         )),
114         encoder.add(layers.ConvLSTM2D(
115             filters=64,
116             kernel_size=(3, 3),
117             strides=(2, 2),
118             padding='same',
119             return_sequences=True,
120             activation='leaky_relu',

```

```

120         kernel_initializer=tf.random_normal_initializer(0., 0.02),
121     )),
122     encoder.add(layers.ConvLSTM2D(
123         filters=64,
124         kernel_size=(3, 3),
125         strides=(2, 2),
126         padding='same',
127         return_sequences=True,
128         activation='leaky_relu',
129         kernel_initializer=tf.random_normal_initializer(0., 0.02),
130     )),
131     encoder.add(layers.ConvLSTM2D(
132         filters=64,
133         kernel_size=(3, 3),
134         strides=(2, 2),
135         padding='same',
136         return_sequences=True,
137         activation='leaky_relu',
138         kernel_initializer=tf.random_normal_initializer(0., 0.02),
139     ))
140     encoder.add(layers.Reshape((30, 64)))
141     encoder.add(layers.LSTM(32, return_sequences=True, activation='leaky_relu'))
142     encoder.add(layers.Conv1D(
143         filters=32,
144         kernel_size=3,
145         strides=8,
146         padding='same',
147         activation='leaky_relu',
148         kernel_initializer=tf.random_normal_initializer(0., 0.02),
149     ))
150     encoder.add(layers.Flatten())
151 else:
152     encoder.add(skip_connection(layers.Conv3D(
153         filters=64,
154         kernel_size=(3, 3, 3),
155         strides=(1, 1, 1),
156         padding='same',
157         activation='leaky_relu',
158         kernel_initializer=tf.random_normal_initializer(0., 0.02),
159     ),
160         (30, 16, 16, 64)
161     ))
162     encoder.add(layers.Conv3D(
163         filters=64,
164         kernel_size=(3, 3, 3),
165         strides=(2, 2, 2),
166         padding='same',
167         activation='leaky_relu',
168         kernel_initializer=tf.random_normal_initializer(0., 0.02),
169     ))
170     encoder.add(layers.Conv3D(
171         filters=64,
172         kernel_size=(3, 3, 3),
173         strides=(2, 2, 2),
174         padding='same',
175         activation='leaky_relu',
176         kernel_initializer=tf.random_normal_initializer(0., 0.02),
177     ))
178     encoder.add(layers.Conv3D(
179         filters=64,
180         kernel_size=(3, 3, 3),
181         strides=(2, 2, 2),
182         padding='same',
183         activation='leaky_relu',
184         kernel_initializer=tf.random_normal_initializer(0., 0.02),
185     ))
186     encoder.add(layers.Conv3D(
187         filters=32,
188         kernel_size=(3, 3, 3),

```

```

189         strides=(1, 2, 2),
190         padding='same',
191         activation='leaky_relu',
192         kernel_initializer=tf.random_normal_initializer(0., 0.02),
193     ))
194     encoder.add(layers.Flatten())
195     encoder.summary()
196
197     # ----- Decoder -----
198     decoder = Sequential([
199         layers.InputLayer(128, ),
200         layers.Dense(30 * 16, activation='leaky_relu'),
201         layers.Reshape((30, 1, 1, 16)),
202         layers.Conv3DTranspose(
203             filters=32,
204             kernel_size=3,
205             strides=(1, 2, 2),
206             padding='same',
207             activation='leaky_relu',
208             kernel_initializer=tf.random_normal_initializer(0., 0.02),
209         ),
210         layers.Conv3DTranspose(
211             filters=64,
212             kernel_size=3,
213             strides=(1, 2, 2),
214             padding='same',
215             activation='leaky_relu',
216             kernel_initializer=tf.random_normal_initializer(0., 0.02),
217         ),
218         layers.Conv3DTranspose(
219             filters=64,
220             kernel_size=3,
221             strides=(1, 2, 2),
222             padding='same',
223             activation='leaky_relu',
224             kernel_initializer=tf.random_normal_initializer(0., 0.02),
225         ),
226         layers.Conv3DTranspose(
227             filters=64,
228             kernel_size=3,
229             strides=(1, 2, 2),
230             padding='same',
231             activation='leaky_relu',
232             kernel_initializer=tf.random_normal_initializer(0., 0.02),
233         ),
234         ConvReshape((30, 16, 16, 64), input_shape, False)
235     ])
236     decoder.summary()
237
238     return Sequential([
239         layers.InputLayer(input_shape=input_shape),
240         encoder,
241         decoder
242     ], name=f'{name_prefix}_LSTM_AE' if lstm else f'{name_prefix}_CNN_AE')
243
244
245 def save_model_json(folder: Path, model: tf.keras.Model) -> None:
246     jsonstr = model.to_json()
247     json_data = json.loads(jsonstr)
248     filename = folder / f'{model.name}.json'
249     with open(filename, 'w') as json_file:
250         json.dump(json_data, json_file, indent=4)
251     print(f'Model save in {filename}')
252     tf.keras.utils.plot_model(
253         model,
254         to_file=f'{model.name}.png',
255         show_shapes=True,
256         show_dtype=False,
257         show_layer_names=True,

```

```

258         expand_nested=True,
259         dpi=400,
260         show_layer_activations=True,
261         show_trainable=False,
262     )
263     plt.show()
264     print(f'Model plot save in {model.name}.png')
265
266 def unet_level_block(
267     level_input_shape: Tuple[int, int, int, int],
268     level_name: str,
269     lstm_layer: bool = False,
270     kernel_size: int = 4,
271     lower_level_block: tf.keras.Model = None,
272     apply_batchnorm: bool = False,
273     apply_dropout: bool = False,
274 ) -> tf.keras.Model:
275     t, w, h, filters_current_level = level_input_shape
276     filters_down_level = lower_level_block.input_shape[-1]
277
278     x_from_upper_level = layers.Input(shape=level_input_shape)
279     if apply_batchnorm:
280         x_from_upper_level_batchnorm = layers.BatchNormalization()(x_from_upper_level)
281     else:
282         x_from_upper_level_batchnorm = x_from_upper_level
283
284     if lstm_layer:
285         x_to_lower_level = layers.ConvLSTM2D(
286             filters=filters_down_level,
287             kernel_size=kernel_size,
288             strides=(2, 2),
289             padding='same',
290             activation='leaky_relu',
291             return_sequences=True,
292             kernel_initializer=tf.random_normal_initializer(0., 0.02),
293             use_bias=False,
294         )(x_from_upper_level_batchnorm)
295     else:
296         x_to_lower_level = layers.Conv3D(
297             filters=filters_down_level,
298             kernel_size=kernel_size,
299             strides=(1, 2, 2),
300             padding='same',
301             activation='leaky_relu',
302             kernel_initializer=tf.random_normal_initializer(0., 0.02),
303             use_bias=False,
304         )(x_from_upper_level_batchnorm)
305
306     if lower_level_block is None:
307         x_from_lower_level = layers.Concatenate()([x_to_lower_level, x_to_lower_level])
308     else:
309         x_from_lower_level = lower_level_block(x_to_lower_level)
310
311     x_up_scaled = layers.Conv3DTranspose(
312         filters=filters_current_level,
313         kernel_size=kernel_size,
314         strides=(1, 2, 2),
315         padding='same',
316         activation='leaky_relu',
317         kernel_initializer=tf.random_normal_initializer(0., 0.02),
318         use_bias=False,
319     )(x_from_lower_level)
320
321     if apply_dropout:
322         x_up_scaled_dropped = layers.Dropout(0.2)(x_up_scaled)
323     else:
324         x_up_scaled_dropped = x_up_scaled
325     x_to_upper_level = layers.Concatenate()([x_up_scaled_dropped, x_from_upper_level])

```

```

327     return tf.keras.Model(x_from_upper_level, x_to_upper_level, name=level_name)
328
329
330 def level_0_block():
331     xin = Input((30, 1, 1, 32))
332     x = layers.Reshape((30, 32))(xin)
333     x = layers.Conv1D(
334         filters=32,
335         kernel_size=16,
336         strides=4,
337         kernel_initializer=tf.random_normal_initializer(0., 0.02),
338     )(x)
339     x = layers.Flatten()(x) #
340     x = layers.Reshape((4, 32))(x)
341     x = layers.Conv1DTranspose(
342         filters=32,
343         kernel_size=18,
344         strides=4,
345         kernel_initializer=tf.random_normal_initializer(0., 0.02),
346     )(x)
347     x = layers.Reshape((30, 1, 1, 32))(x)
348     x_out = layers.Concatenate()([xin, x])
349     return tf.keras.Model(xin, x_out)
350
351
352 def cnn_unet_model(input_shape: Tuple[int, int, int, int], name_prefix: str = '') -> tf.
                                     keras.Model:
353     level_0 = level_0_block()
354
355     level_1 = unet_level_block(
356         level_name='CNN_Unet_level_1',
357         level_input_shape=(30, 2, 2, 32),
358         lower_level_block=level_0,
359         apply_dropout=True,
360     )
361
362     level_2 = unet_level_block(
363         level_name='CNN_Unet_level_2',
364         level_input_shape=(30, 4, 4, 64),
365         lower_level_block=level_1,
366     )
367
368     level_3 = unet_level_block(
369         level_name='CNN_Unet_level_3',
370         level_input_shape=(30, 8, 8, 64),
371         lower_level_block=level_2)
372
373     level_4 = unet_level_block(
374         level_name='CNN_Unet_level_4',
375         level_input_shape=(30, 16, 16, 64),
376         lower_level_block=level_3,
377         apply_batchnorm=True,
378     )
379
380     return Sequential([
381         layers.InputLayer(input_shape=input_shape),
382         ConvReshape(
383             input_shape=input_shape,
384             output_shape=(30, 16, 16, 64),
385             lstm=False
386         ),
387         level_4,
388         ConvReshape(input_shape=(30, 16, 16, 128), output_shape=input_shape, lstm=False),
389     ], name=f'{name_prefix}_CNN_Unet_AutoEncoder')
390
391
392 def lstm_unet_model(input_shape: Tuple[int, int, int, int], name_prefix: str = '') -> Model:
393     level_0 = level_0_block()
394

```

```

395     level_1 = unet_level_block(
396         level_name='LSTM_Unet_level_1',
397         level_input_shape=(30, 2, 2, 32),
398         lower_level_block=level_0,
399         lstm_layer=True,
400         apply_dropout=True,
401     )
402
403     level_2 = unet_level_block(
404         level_name='LSTM_Unet_level_2',
405         level_input_shape=(30, 4, 4, 64),
406         lower_level_block=level_1,
407         lstm_layer=True,
408     )
409
410     level_3 = unet_level_block(
411         level_name='LSTM_Unet_level_3',
412         level_input_shape=(30, 8, 8, 64),
413         lower_level_block=level_2,
414         lstm_layer=True,
415     )
416
417     level_4 = unet_level_block(
418         level_name='LSTM_Unet_level_4',
419         level_input_shape=(30, 16, 16, 64),
420         lower_level_block=level_3,
421         lstm_layer=True,
422         apply_batchnorm=True,
423     )
424
425     return Sequential([
426         layers.InputLayer(input_shape),
427         ConvReshape(
428             input_shape=input_shape,
429             output_shape=(30, 16, 16, 64),
430             lstm=False
431         ),
432         level_4,
433         ConvReshape(input_shape=(30, 16, 16, 128), output_shape=input_shape, lstm=True),
434     ], name=f'{name_prefix}_LSTM_Unet_AutoEncoder')
435
436
437 if __name__ == '__main__':
438     input_shape = (30, 1, 1, 23)
439     name_prefix = 'RealData23S1CC'
440     models_folder = Path(__file__).parent / 'models'
441     models_folder.mkdir(parents=True, exist_ok=True)
442
443     # CNN-AE AutoEncoder
444     model = autoencoder(input_shape, False, name_prefix)
445     save_model_json(models_folder, model)
446
447     # LSTM-AE AutoEncoder
448     model = autoencoder(input_shape, True, name_prefix)
449     save_model_json(models_folder, model)
450
451     # U-net CNN AutoEncoder
452     model = cnn_unet_model(input_shape, name_prefix)
453     save_model_json(models_folder, model)
454
455     # U-net LSTM AutoEncoder
456     model = lstm_unet_model(input_shape, name_prefix)
457     save_model_json(models_folder, model)

```

ملخص

نستكشف هذه الرسالة العلمية تطبيق تقنيات الذكاء الاصطناعي (Artificial Intelligence - AI) ومعالجة دوال الإشارات (SP) في سياق دمج المستشعرات المطبقة في تجربة فيزياء الجسيمات عالية الطاقة في مصادم الهادرونات الكبير (LHC) في المركز الأوروبي للأبحاث النووية (CERN). الهدف الرئيسي هو المساعدة في الحفاظ على سلامة عمليات جمع البيانات من مجموعة هائلة من المستشعرات المستخدمة في كاشف توقيت الجسيمات ذات الأيونات الأقل (MTD) الموجود في الطبقات الداخلية للكاشف المغناطيس الحلزوني المدمج (CMS) وتطوير منصة مراقبة سهلة الاستخدام لجمع وعرض الكم الهائل من بيانات المستشعرات بشكل مدمج لإستخدام المراقب. لذلك، يهدف جزء كبير من هذا العمل إلى اكتشاف وإطلاق تحذيرات عن المشكلات العملية المحتملة قبل أن تتفاقم من خلال تطبيق تقنيات الذكاء الاصطناعي المختلفة للكشف عن الأعطال العملية، ويتم ذلك عن طريق شبكات العصبية التلقائية (autoencoder) بما في تلك الشبكات العصبية الالتفافية (CNN) والشبكة الالتفافية ثنائية الأبعاد مع شبكة عصبية متكررة طويلة الامد قصيرة الذاكرة (Conv2DLSTM) و CNN مع بنية شبكة العصبية بشكل الحذوه (U-net)، و Conv2DLSTM مع بنية شبكة العصبية بشكل الحذوه (U-net). وقد تبين أن الشبكة العصبية الأخيرة توفر أفضل أداء بدقة 96.2% على البيانات المحاكاة. تشير المحاكاة الحاسوبية إلى أن هذه الخوارزمية قوية جداً ضد الضوضاء المضافة حيث تبين أن أدائها تراجع بنحو 1% فقط مع انخفاض نسبة الإشارة إلى الضوضاء (SNR) من 17dB إلى 5dB. تم التحقيق في خوارزمية إزالة الضوضاء بالمويجات (wavelet denoising) للتخفيف من تأثير الضوضاء نظراً للبيئة التشغيلية القاسية في CMS بنجاح محدود، حيث في إشارات المستشعرات ذات SNR المنخفضة تسببت في إخفاء بيانات التي تشير إلى الأعطال العملية وبالتالي تدنى أداء الخوارزمية. كان هناك صعوبة في الحصول على بيانات المستشعرات و لذلك و في وقت كتابة هذا العمل تم الحصول على بيانات جزئية فقط. أظهرت النتائج الأولية لتطبيق الخوارزمية على البيانات الحقيقية الجزئية التي تم الحصول عليها دقة مقارنة تبلغ 94.5%. بالإضافة إلى مساهمات في خدمات MTD المطورة في هذا العمل باستخدام مكتبة FastAPI بلغة Python وإطار عمل xDAQ بلغة C++ حيث يمكن وضع كاشف الأعطال العملية المقترحة فيه للمساهمة والمحافظة على تشغيل عمليات إستلام بيانات MTD بمثابة.



جامعة البحرين
كلية الهندسة
قسم الهندسة الكهربائية والإلكترونية

الرسالة العملية للماجستير

دمج المستشعرات باستخدام معالجة الإشارات وأدوات الذكاء الاصطناعي المطبقة على كاشف CERN-CMS

كتب بواسطة
حسن حيدر الجامع

بإشراف
د. زهير البحري
د. محمد أوزغور شاهين

أطروحة مقدمة لاستيفاء متطلبات درجة ماجستير العلوم في نظام الذكاء الاصطناعي

19th May 2024