# Appendices

[Appendix A: Plotting the Bethe-Bloch Equation 111](#_Toc14519599)

[Appendix B: Plotting Binary Cross-Entropy 118](#_Toc14519600)

[Appendix C: The Anatomy Of An AliROOT Analysis Task 119](#_Toc14519601)

[Appendix D: Software Environment, Packages & Utilities 124](#_Toc14519602)

[Appendix E: Running and Monitoring Root Analysis Tasks 126](#_Toc14519603)

[Appendix F: Data Extraction, Data Quality and Data Pre-processing 130](#_Toc14519604)

[Appendix G: Old Methods Section 132](#_Toc14519605)

[Appendix H: Old Results Section 136](#_Toc14519606)

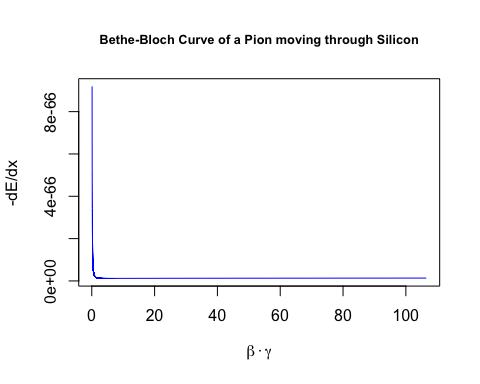
Appendix A: Plotting the Bethe-Bloch Equation

Create a Bethe-Bloch function:

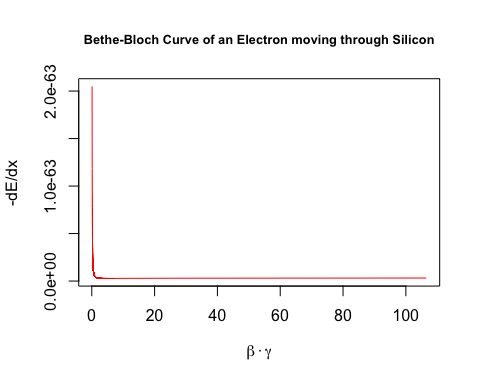
#Planck's constant:  
h <- 6.62607004e-34  
  
#Speed of light m/s  
c <- 299792458  
  
#Fine structure constant  
alpha <- 1/137  
  
#Mass of an electron Mass/GeV  
  
m.e <- 0.005  
  
#Density n, atomic number Z, the fraction of the speed of light the particle is moving at, beta, and the particle's velocity v are specified as parameters to the equation  
  
  
dE.dx <- function(n,Z,v,beta){  
 -4 \* pi \* h^2 \* c^2 \* alpha^2 \* ((n \* Z)/(m.e \* v^2)) \* log(((2 \* beta^2 \* gamma^2 \* c^2 \* m.e)/(I.e)) - beta^2,base=exp(1))  
}  
  
#For an electron traversing a silicon detector:  
  
v <- seq(0.1\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
  
  
Z <- 14  
  
#Effective ionization potential of the material  
  
I.e <- 10 \* Z  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
require(latex2exp)

## Loading required package: latex2exp

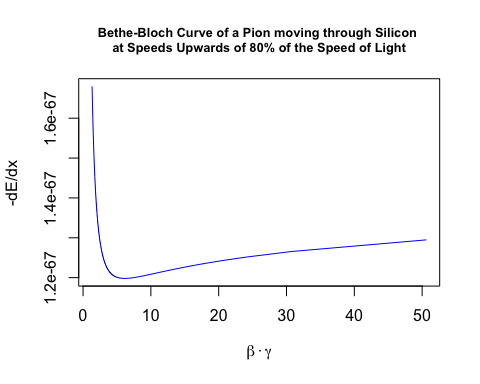
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



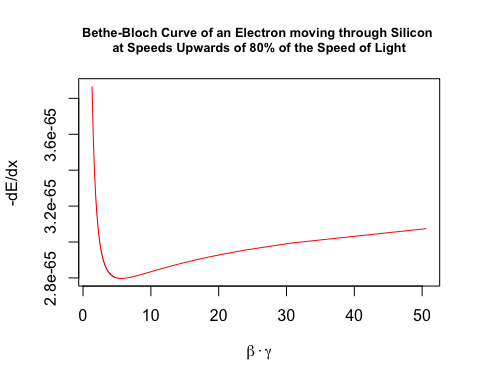
plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



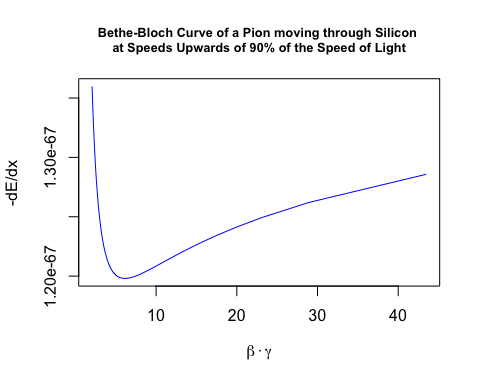
v <- seq(0.8\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
m.e <- 0.005  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon \nat Speeds Upwards of 80% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



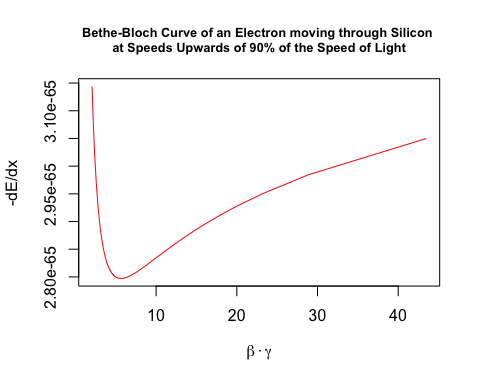
plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon \nat Speeds Upwards of 80% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



v <- seq(0.9\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
m.e <- 0.005  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon \nat Speeds Upwards of 90% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon \nat Speeds Upwards of 90% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



Appendix B: Plotting Binary Cross-Entropy

Define a function to plot the binary cross-entropy loss function:

cross.entropy <- function(y,p){  
 -(y \* log(p,base = 10) + ((1-y)\*(1 - log(p,base=10))))  
}  
  
#if the predicted class is 1:  
  
y <- 1  
  
p <- seq(0,1,0.01)  
  
loss <- cross.entropy(y,p)  
  
require(latex2exp)

## Loading required package: latex2exp

plot(x=p,y=loss, type="b", col=rainbow(250),cex=0.5, main = TeX("J($\\theta$) = -(y log(p)-(1-log(p)))"),ylab = "Cross Entropy", xlab = TeX("$\\hat{y}$"))



Appendix C: The Anatomy Of An AliROOT Analysis Task

In AliROOT, all analysis tasks inherit from the base class **AliAnalysisTaskSE** (where SE stands for Single Event), which in turn is derived from the base class **AliAnalysisTask.**

All analysis tasks done in AliROOT inherit the following base methods from **AliAnalysisTaskSE**:

AliAnalysisTaskSE::AliAnalysisTaskSE();//constructor1

AliAnalysisTaskSE::AliAnalysisTaskSE(const char\*);//constructor2

AliAnalysisTaskSE::~AliAnalysisTaskSE();//destructor

AliAnalysisTaskSE::UserCreateOutputObjects();//user-defined output objects (results of physics analyses, which can be attached to output files)

AliAnalysisTaskSE::UserExec(Option\_t\*);//event loop, called for each event in the analysis: checks conditions for inclusion, accesses physics objects, fills histograms or other data containers with attributes from event

AliAnalysisTaskSE::Terminate(Option\_t\*); //deallocates memory after all steps in analysis have completed

The final element of an analysis task in AliROOT is the (.C) macro file, which creates and configures an instance of the particular C++ class.

##### The Class Header (.h)

Reproduced and modified from (35):

#ifndef AliAnalysisTaskMyTask\_H //include guard (aids in prevention of double inclusion, which may result from including parent and child classes, leading to multiple definitions for class members)

#define AliAnalysisTaskMyTask\_H //part of include guard

class AliAnalysisTaskMyTask : public AliAnalysisTaskSE //we define a class AliAnalysisTaskMyTask, which inherits from the base class AliAnalysisTaskSE

{

public:

// two class constructors, called when a new instance of the class is created

AliAnalysisTaskMyTask();

AliAnalysisTaskMyTask(const char \*name);

// class destructor, called when this instance of the class is deleted

virtual ~AliAnalysisTaskMyTask();

// called once at beginning of runtime

virtual void UserCreateOutputObjects();

// called for each event

virtual void UserExec(Option\_t\\* option);

// called at end of analysis

virtual void Terminate(Option\_t\\* option);

//class members

private:

AliAODEvent\* fAOD; //!<! pointer to a single input event

TList\* fOutputList; //!<! pointer to an output list, which holds all the output objects of the analysis

TH1F\* fHistPt; //!<! pointer to a histogram containing the transverse momentum (Pt) spectrum

//note that the !<! expression above is seen and evaluated by ROOT and is used in the generation of ROOT documentation

//ClassDef definition:

/// \cond CLASSDEF //surrounding comments for documentation generation

ClassDef(AliAnalysisTaskMyTask, 1); //this is a C pre-processor macro, used when class derives from TObject: it contains member declarations and inserts a few new members into the class, version number is incremented from 1 when definition of class changes

/// \endcond}; //surrounding comments for documentation generation

#endif //part of include guard

##### The Class Implementation (.cxx)

Reproduced and modified from (35):

//include statements for UserCreateOutputObjects:

**#include "TList.h"** *//TList class, an instance of which will contain a histogram in this example*

#include "TH1F.h" //ROOT 1-dimensional histogram class with one float per channel

//include statement for UserExec:

#include "AliAODEvent.h"

//implementation of class constructors:

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask() : AliAnalysisTaskSE(),

//members of the class are initialized in the constructors with their default values, if default values are not specified, these will be filled with random values, which could lead to unexpected behaviour

fAOD{0}, fOutputList{0}, fHistPt{0}

{

// This first constructor is the ROOT IO constructor, memory should not be allocated here

}

//in the second constructor, below, the input and output objects handled by the class are defined

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char\* name) : AliAnalysisTaskSE(name),

fAOD{0}, fOutputList{0}, fHistPt{0}

{

//input object is a TChain

DefineInput(0, TChain::Class());

//output object is a TList

DefineOutput(1, TList::Class());

}

//implementation of the UserCreateOutputObjects class:

AliAnalysisTaskMyTask::UserCreateOutputObjects()

{

// create a new TList that OWNS its objects

fOutputList = new TList();

fOutputList->SetOwner(true);

// create a histogram:

//from ROOT’s online documentation, this is the constructor for a TH1F:

//TH1F (const char \*name, const char \*title, Int\_t nbinsx, Double\_t xlow, Double\_t xup)

//seen below, we give the histogram the pointer name defined in the header file and give the histogram plot the same title, we define the histogram itself to have 100 bins on an x-axis bounded by [0,100]

fHistPt = new TH1F("fHistPt", "fHistPt", 100, 0, 100);

//add the histogram to the output list:

fOutputList->Add(fHistPt);

// add the list to our output file

PostData(1,fOutputList); //calling PostData() notifies client tasks of the fOutPutList data container that its contents have changed

}

//UserExec: the “event loop” (operations defined here are called for each event in the analysis):

AliAnalysisTaskMyTask::UserExec(Option\_t\*)

{

// get an input event from the analysis manager and cast it as an AliAODEvent

fAOD = dynamic\_cast<AliAODEvent\*>(InputEvent());

// check if there actually is an event, and throw a fatal exception with error message if not

if(!fAOD)

::Fatal("AliAnalysisTaskMyTask::UserExec", "No AOD event found, check the event handler.");

// Loop over all the tracks in the event and fill the histogram

// get the number of tracks in the input event

int iTracks{fAOD->GetNumberOfTracks()};

// iterate through all the tracks in the event:

for(int i{0}; i < iTracks; i++) {

//get the current track, cast it as an AliAODTrack

AliAODTrack\* track = static\_cast<AliAODTrack\*>(fAOD->GetTrack(i));

//if the track variable does not exist after the above operation, continue to the next iteration of the loop

if(!track) continue;

// here we do some track selection

if(!track->TestFilterbit(128) continue;

// get the transverse momentum of the track and fill the histogram with this data

fHistPt->Fill(track->Pt());

}

// save the output list

PostData(1, fOutputList);

}

##### The AddTask macro (.C)

Reproduced and modified from (35):

//this file instantiates our class, defines its input and output, and connects it to the analysis manager

AliAnalysisTaskMyTask\* AddMyTask(TString name = "name") {

//get a pointer to the analysis manager

AliAnalysisManager \*mgr = AliAnalysisManager::GetAnalysisManager();

// resolve the name of the output file

TString fileName = AliAnalysisManager::GetCommonFileName();

fileName += ":MyTask"; // create a subfolder in this file

// create an instance of the analysis task

AliAnalysisTaskMyTask\* task = new AliAnalysisTaskMyTask(name.Data());

// add this task to the analysis manager

mgr->AddTask(task);

// connect the manager to the task’s input container

mgr->ConnectInput(task,0,mgr->GetCommonInputContainer());

// connect the manager to the task’s output container (TList)

mgr->ConnectOutput(task,1,mgr->CreateContainer("MyOutputContainer", TList::Class(), AliAnalysisManager::kOutputContainer, fileName.Data()));

// important: return a pointer to this task

return task;

}

Appendix D: Software Environment, Packages & Utilities

## Software Environment

### AiROOT

AliROOT was built locally using alidock Docker container.

AliROOT was built from source on the hep01 server hosted at UCT

### R Statistical Software

#### Packages

### ROOTR

### Keras & Tensorflow

### Utilities

#### Makefiles

all: gridfiles.md5

gridfiles.xml: query.sh

./$< > $@

gridfiles.md5: gridfiles.xml

xsltproc /alice/data/util/xml2md5.xsl $< > $@

download: $(shell cut -c 49- files.md5)

/alice/data/%:

mkdir -p $(dir $@)

alien\_cp alien:$@ file:$@

#### User specified aliases in ~/.bashrc

# User specific aliases and functions

alias initialize\_aliroot='/cvmfs/alice.cern.ch/bin/alienv enter VO\_ALICE@AliPhysics::vAN-20180902-1'

alias my\_alice='alienv -w /alice/gviljoen/alice/sw enter VO\_ALICE@AliPhysics::latest'

#### Remote Editing

#### Rsync

rsync -av --stats --progress --include="\*/" --include="\*.txt" --exclude="\*.C" --exclude="\*.cxx" --exclude="\*.h" --exclude="\*.root" --exclude="\*.ps" --exclude="\*.d" --exclude="\*.so" --exclude "\*.proc" gviljoen@hep01.phy.uct.ac.za:/alice/gviljoen/trdpid/adj\_sim/test .

X11 Forwarding

Atom packages:

* Remote Atom Server
* PlatformIO-IDE-Terminal

##### Killing a process being listened to on the remote port 52698:

List processes that are owned by me:

ps aux | grep gviljoen

Find the sshd process being listened to on port 52698 and kill it, by running:

kill -9 $processid

In this case, here is the suspect process ($processid = 28525):

gviljoen 28525 0.0 0.0 119612 2168 ? S 13:02 0:00 sshd: gviljoen@pts/0

### Python

Appendix E: Running and Monitoring Root Analysis Tasks

Once one is happy with the analysis task defined, one first needs to enter AliPhysics, by using one of the user-defined aliases, e.g.:

initialize\_aliroot

Then, one gets a token from alien, to access the grid. This token will be valid for 24 hours. Since my CERN username is not the same as my username on HEP01, the command is:

alien-token-init username

Once the above commands have been run, one can run the analysis task on the grid, by setting the following parameters in the analysis macro (ana.C):

Bool\_t local = kFALSE;

Bool\_t gridTest = kFALSE;

Adding the appropriate run number and output directory:

alienHandler->AddRunNumber(265377);

alienHandler->SetGridWorkingDir("new-wd-momentum-test");

alienHandler->SetGridOutputDir("outDir265378");

Setting the run mode and starting the analysis:

alienHandler->SetRunMode("full");

//alienHandler->SetRunMode("terminate"); //this is run for merging stages

mgr->StartAnalysis("grid");

Assuming that one has added the appropriate CERN certificates, one can then view, manage and download the output of one’s jobs on the MonALISA grid monitoring site for ALICE see Figure 39 for an example screenshot of user job monitoring and Figure 40 for the user interface for viewing the directory structure for the ALICE grid, in particular the user’s working directory:

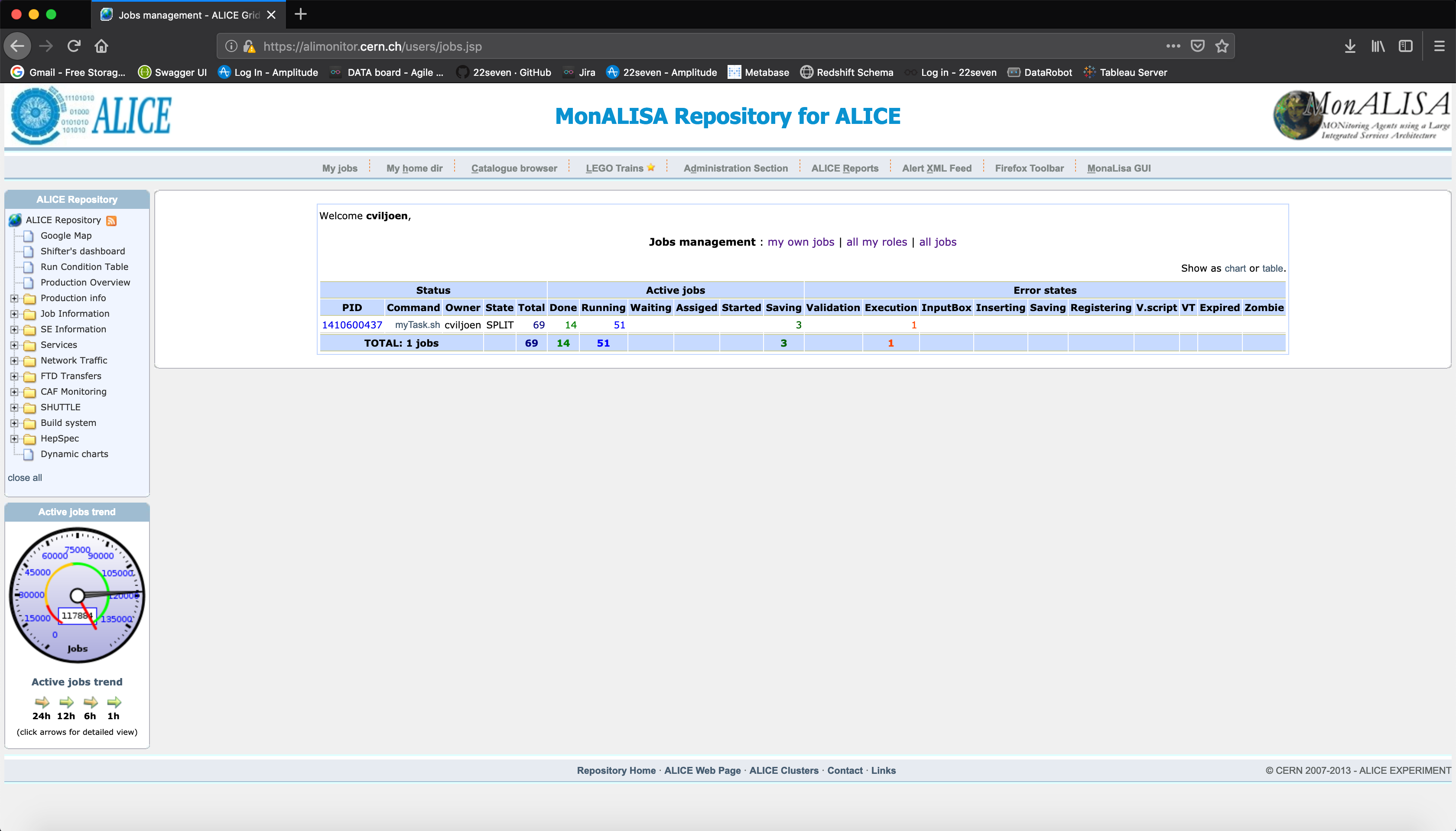


Figure : MonALISA Alice grid monitoring site, user jobs at url: <https://alimonitor.cern.ch/users/jobs.jsp>

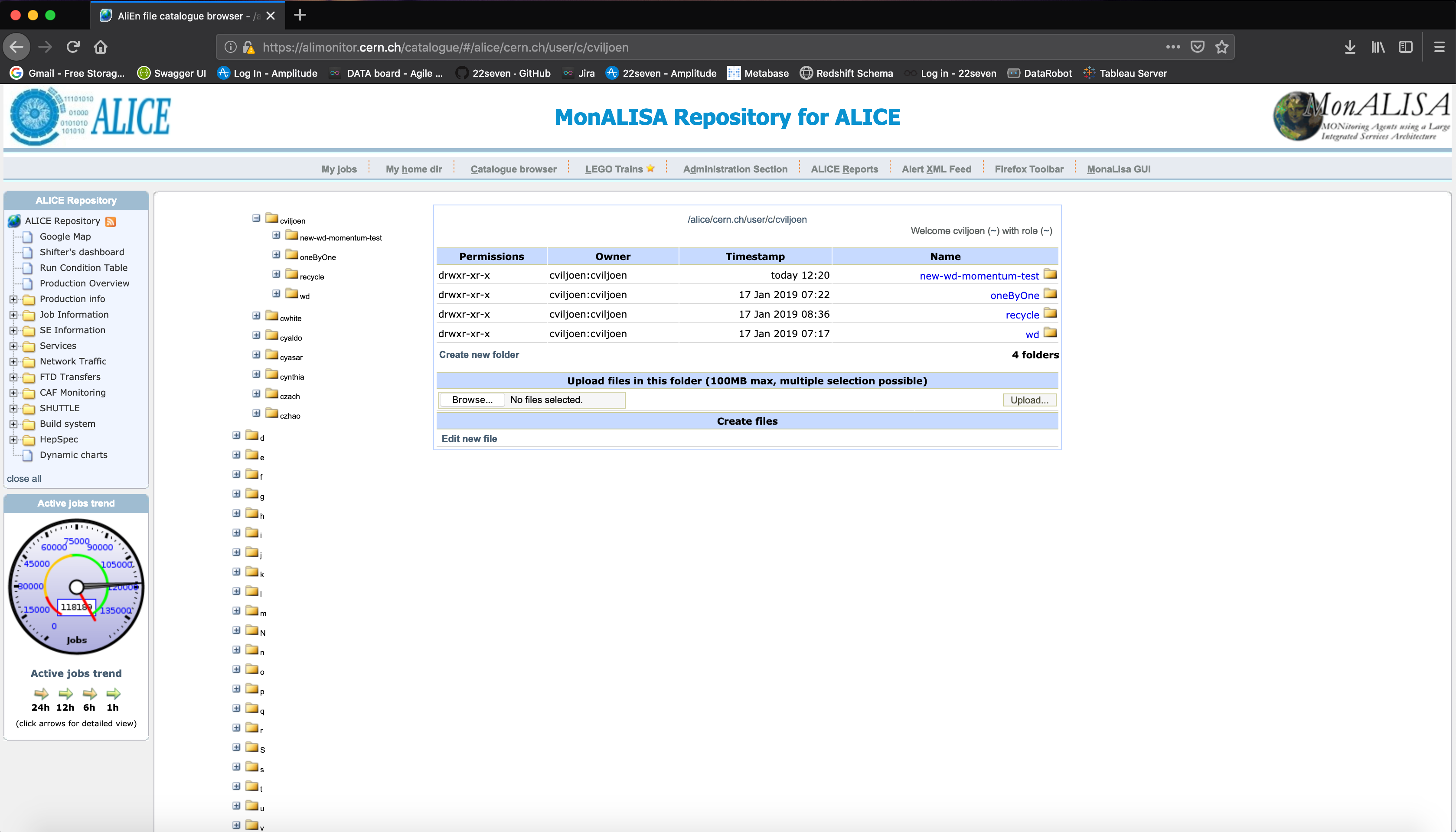


Figure : User working directory structure on MonALISA at url: <https://alimonitor.cern.ch/catalogue/#/alice/cern.ch/user/c/cviljoen>

In Figure 41, a screenshot shows how subjobs belonging to a masterjob can be tracked by clicking on the process ID on the MonALISA jobs management webpage:

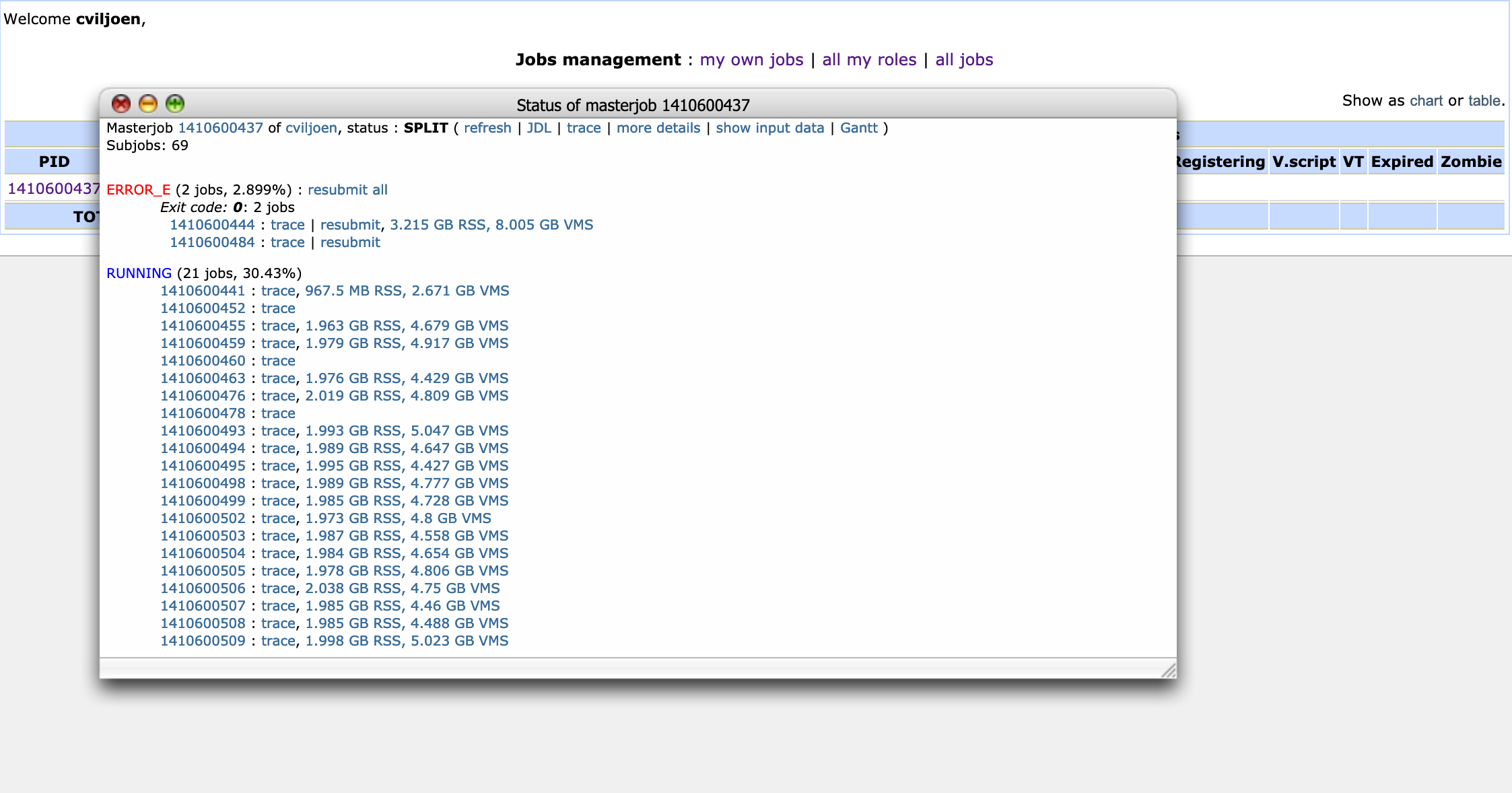


Figure : Tracking the status of subjobs of a master-job, by clicking on the process id (PID)

One can resubmit errored subjobs by browsing through the various error states in the “Status of masterjob” view and clicking on “resubmit all” for all processes that are in a specific error state.

The trace of a subjob (see Figure 42 for an example screenshot) can give hints as to what caused a specific subjob to fall into an error state. In this case the job has an error state “ERROR\_E”, i.e. “Error in Execution”, since the job is using too much memory (memory and storage limits are allocated to each user and overusing either can downgrade the priority of a user’s jobs).

The alien shell can be accessed by running

aliensh

This gives access to the alien terminal, which is not strictly a bash terminal, but has similar commands, for instance the shell command to forcefully and recursively remove a directory:

rm -rf directory

would be achieved on an alien terminal by running:

rmdir directory

Killing a job is done in a similar fashion to the normal shell workflow, i.e. running

ps

To list the currently active processes and

kill $(process-id)

To kill a process and its attendant subprocesses, in case you figured out that you made a mistake and want to terminate a running process early for whatever reason.

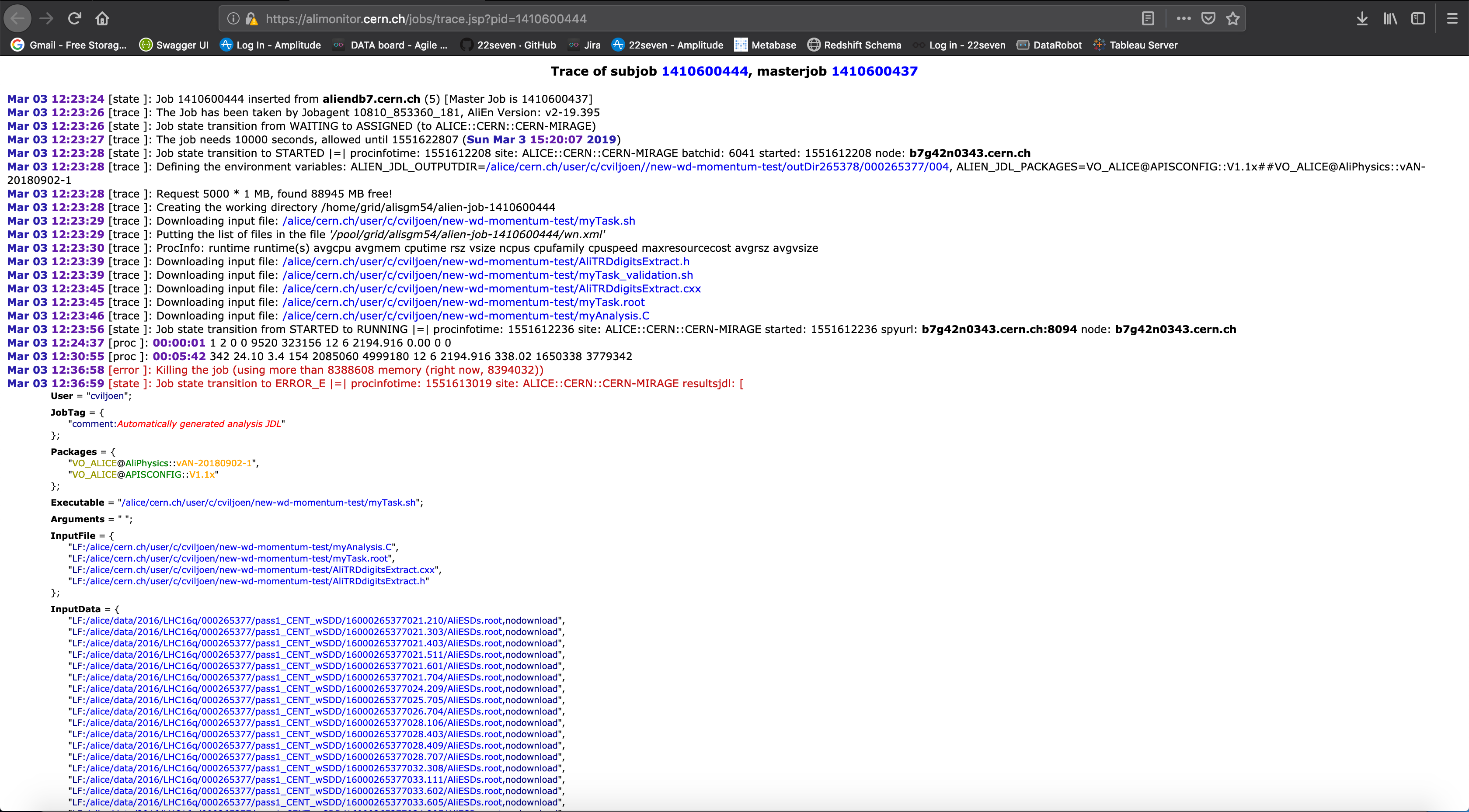


Figure : Example trace of a subjob on MonALISA

Appendix F: Data Extraction, Data Quality and Data Pre-processing

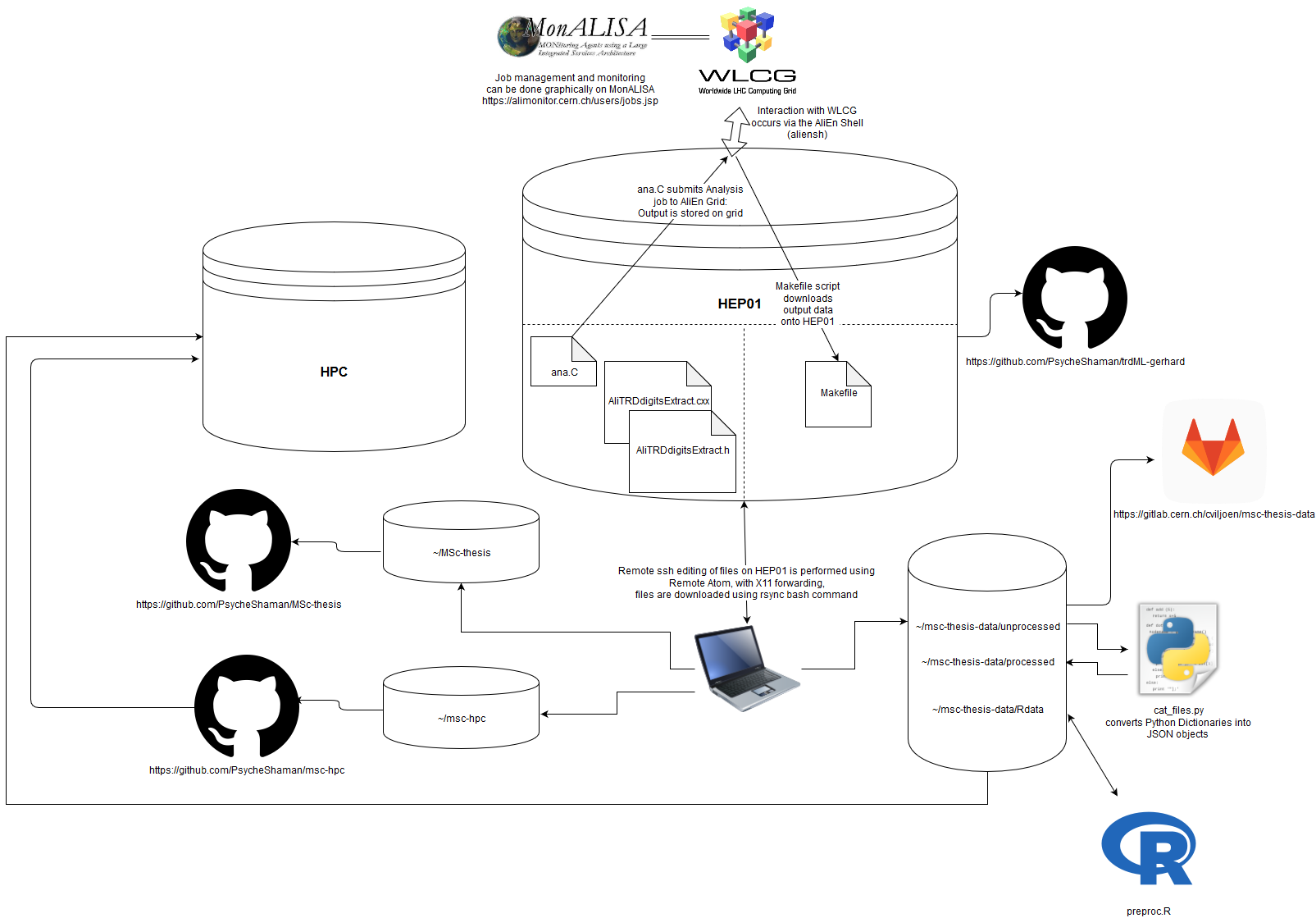


Figure : Data pre-processing environment and repository logic

### Running Digits Extract Task on AliEn Grid

* Stage 1
* Stage 2
* Stage 3

### Data Extraction from WLCG

#### From Alien to HEP01

Makefile

#### From HEP01 to Local Machine

Into data backup directory: <https://gitlab.cern.ch/cviljoen/msc-thesis-data>

Directory structure

scp -r gviljoen@hep01.phy.uct.ac.za:/alice/cern.ch/user/c/cviljoen/wd/outDir265377/000265377/ .

rsync -av --stats --progress gviljoen@hep01.phy.uct.ac.za://alice/cern.ch/user/c/cviljoen/wd/od/ .

#### From Local Machine to HPC

### Data Quality Assessment and Descriptive Statistics

### File Merging and Conversion of Python Dictionaries to JSON Objects

Cat\_files.py

### Loading JSON Files into R Environment and Data Wrangling for Deep Learning

Wrangle.R

### UCT HPC Cluster

Compiler variables set in ~/.R/Makevars

CC = gcc -std=gnu99

Install.packages command needs to be modified to write packages in a directory where there are permissions and where the CRAN mirror is set, dependencies=TRUE allows R to read the Makevars compiler variables.

install.packages(pkgs="keras",lib="/scratch/username",

repos="https://cloud.r-project.org",dependencies=TRUE)

Appendix G: Old Methods Section

#### Model 1

An initial benchmark feedforward model was built, compiled and trained, according to the following lines of Python code:

model1 = Sequential([

Dense(256, input\_shape=(24,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(64),

Activation('relu'),

Dense(2),

Activation('softmax')

])

model1.compile(loss='categorical\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

history = model1.fit(x\_train, y\_train,

epochs=epochs,

validation\_split=0.15,

shuffle=True,

verbose=2)

The input features to this model were as follows:

* Time-bin sums across all pads, divided by the mean of the entire x sample
* Missing data removed
* Electrons oversampled (the electron sample in the training data was taken thrice)

This model was trained on 5778261 samples and validated on 642029 samples.

As can be seen in 0, this model failed to train, so an approach was taken to account for class imbalances using a different method, and by starting with a very simple architecture and sequentially adding complexity to the model.

#### Sequential Model Building

##### Stage 1

All electron tracks were included and a pion sample twice the size of the electron sample was added, before partitioning data into a training and test set.

Class imbalances were accounted for by allowing error in electron classification to contribute proportionately more to the loss function.

As per the approach followed by the current classification neural network in production at the TRD, timebins were compressed by summing across three timebins at a time, to create 8 new features, which were added to the 24 timebins already available to the neural network as input features.

Class weights were accounted for as follows:

class\_weights = class\_weight.compute\_class\_weight('balanced',

np.unique(y\_train),

y\_train)

class\_weights = {0:class\_weights[0],1:class\_weights[1]}

The model was built, compiled and trained according to the following Python code:

sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)

model1\_dropout\_0\_5 = Sequential([

Dense(32, input\_shape=(32,)),

Activation('relu'),

Dense(2),

Activation('softmax')

])

batch\_size=32

model1\_dropout\_0\_5.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

history = model1\_dropout\_0\_5.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_split=0.1,

shuffle=True,

verbose=2,

class\_weight=class\_weights)

Model was trained on 883591 samples, and validated on 98177 samples, due to undersampling of pions.

As can be seen in 0, this model did train, in contrast to the first model, although it did not achieve validation accuracy above 75%.

##### Stage 2

Following the successful running of the above model, the same model was run, without compensating for class imbalances by undersampling pions, but by maintaining the proportionally greater contribution to the loss function by the underrepresented “electron” class.

This model was trained on 2720444 samples and validated on 302272 samples.

##### Stage 3

After successful training of the single layer, 32 node neural network above, a larger network was constructed as follows, using the same dataset, optimizer, etc.

model1\_dropout\_0\_5 = Sequential([

Dense(128, input\_shape=(32,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(2),

Activation('softmax')

])

As can be seen in 0, increasing the model capacity in this way does have its benefits in terms of accuracy, without seeming to overfit too much, therefore the next model was built with much higher capacity.

##### Stage 4

model1\_dropout\_0\_5 = Sequential([

Dense(128, input\_shape=(32,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(2),

Activation('softmax')

])

While this model was slightly more accurate than those discussed before, it is clear when looking at 0 that the model was not generalizable to the validation set, therefore, before increasing model complexity, regularization in the form of dropout was introduced as follows:

### 2D Convolutional Neural Networks

#### Stage 1

Trained on 770981 samples, validated on 85655 samples:

epochs = 100

model = Sequential()

model.add(Conv2D(16, (2, 2), padding='valid',input\_shape=(17,24,1),data\_format="channels\_last"))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(3, 3)))

model.add(Flatten())

model.add(Dense(256))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(2))

model.add(Activation('softmax'))

sgd = tensorflow.keras.optimizers.SGD(lr=0.01, clipvalue=0.5)

model.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

history=model.fit(x\_train, y\_train,

epochs=epochs,

validation\_split=0.1,

shuffle=True)

#### Stage 2

epochs = 100

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu', input\_shape=(x\_train.shape[1],x\_train.shape[2],x\_train.shape[3]),data\_format="channels\_last"))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

sgd = tensorflow.keras.optimizers.SGD(lr=0.01, clipvalue=0.5)

model.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

batch\_size=32

history=model.fit(x\_train, y\_train,

epochs=epochs,

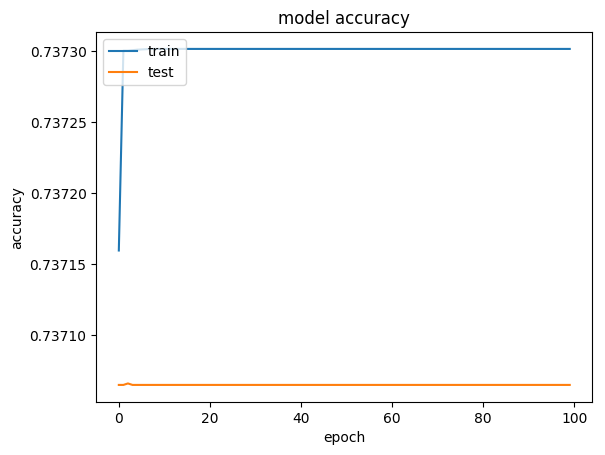
validation\_split=0.1,

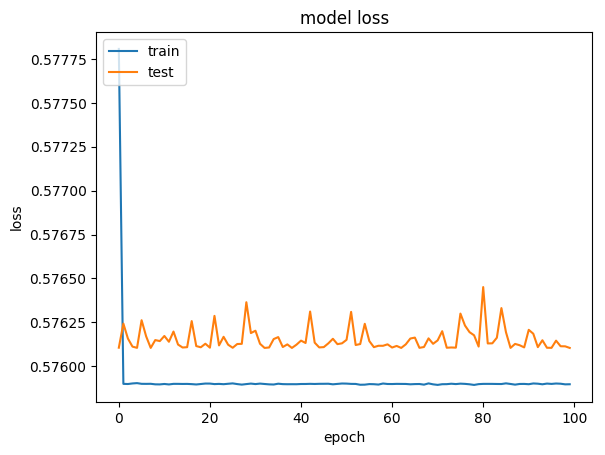
shuffle=True)

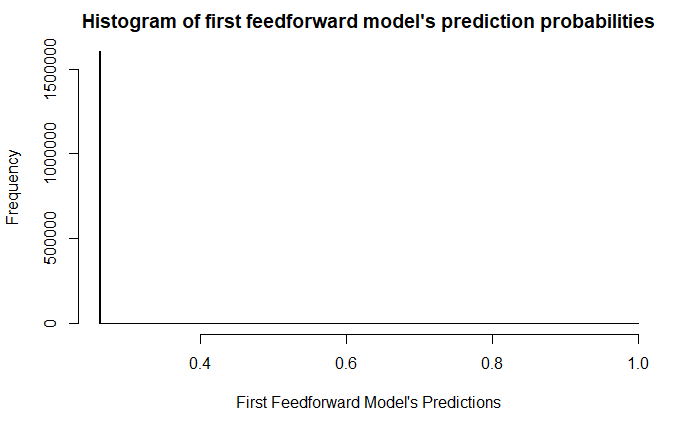
### Recurrent Neural Networks

Appendix H: Old Results Section

#### Model 1

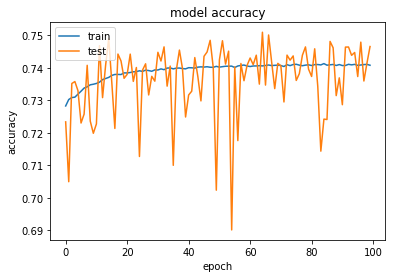


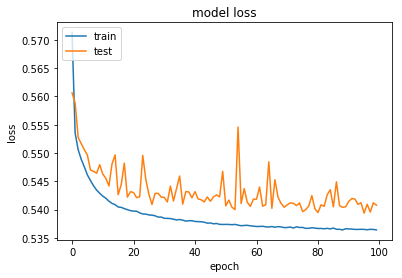


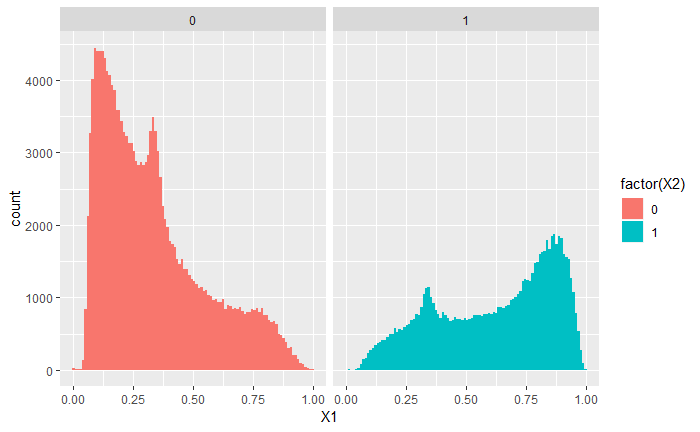


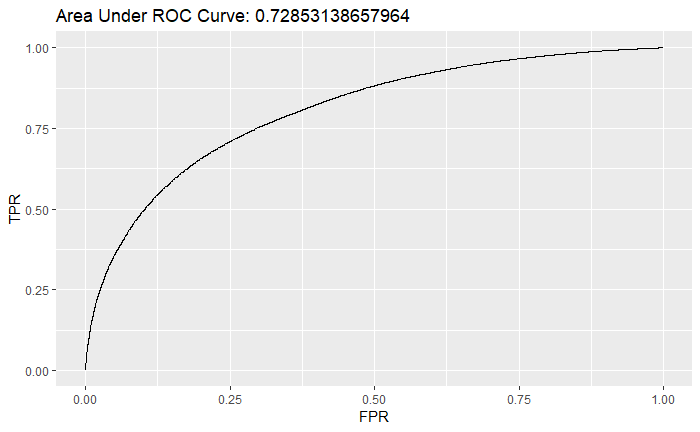
#### Sequential Model Building

##### Stage 1

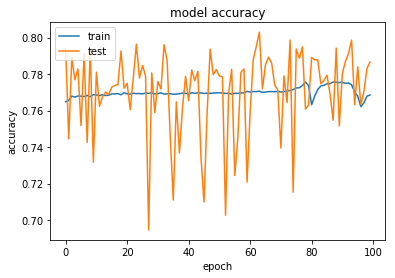


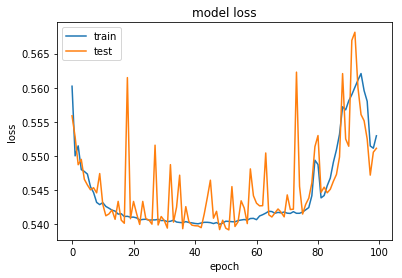


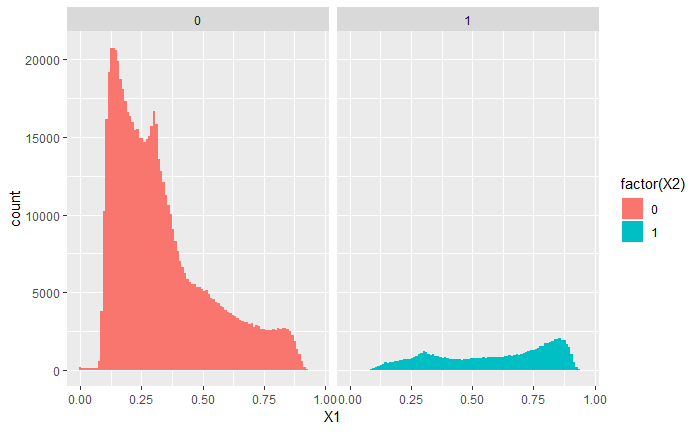


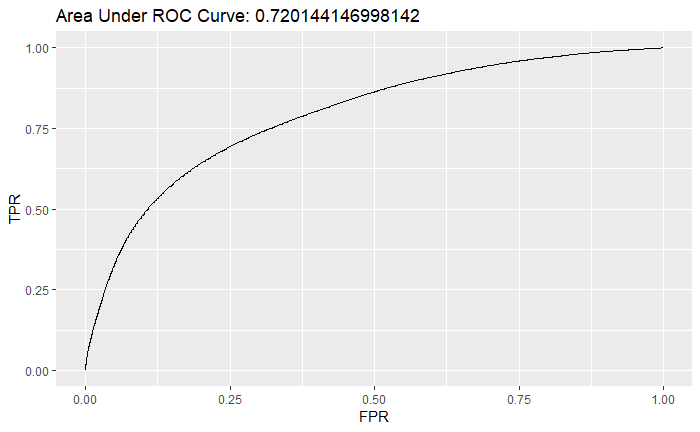


##### Stage 2

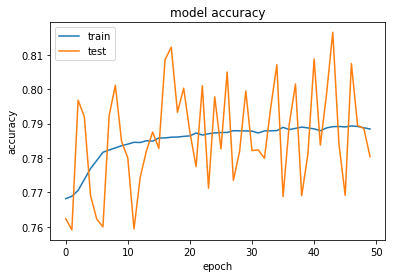


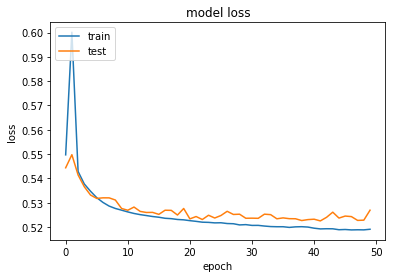


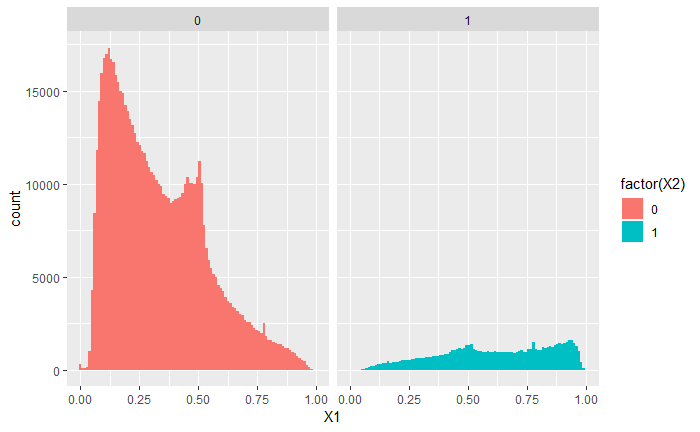


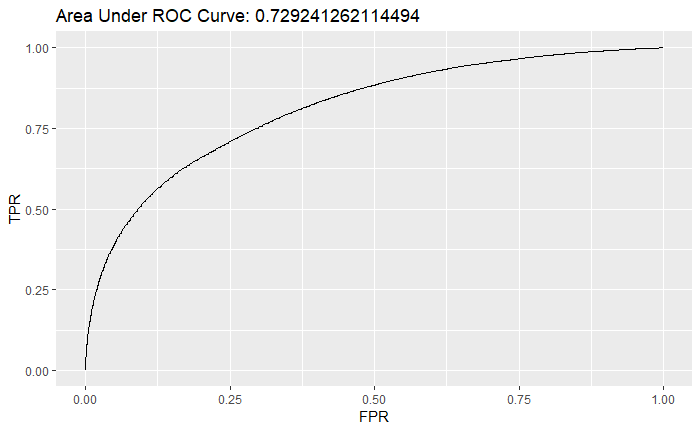


##### Stage 3

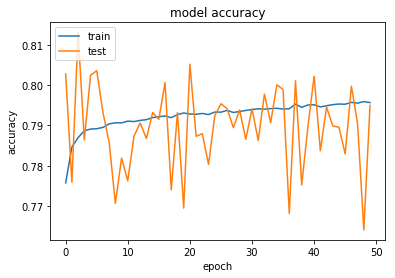


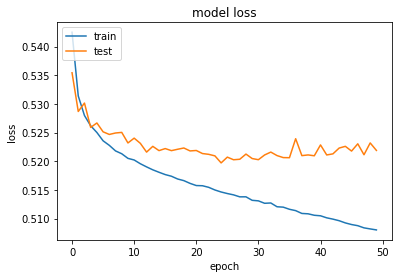






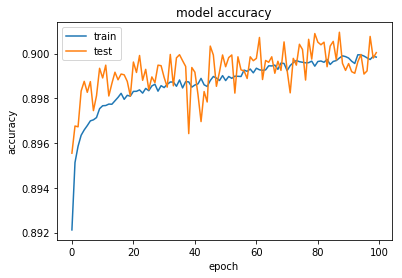
##### Stage 4

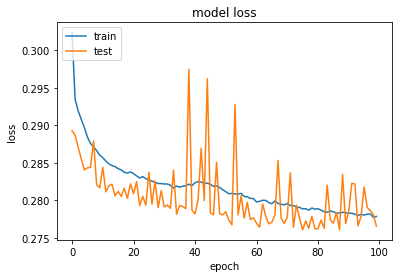




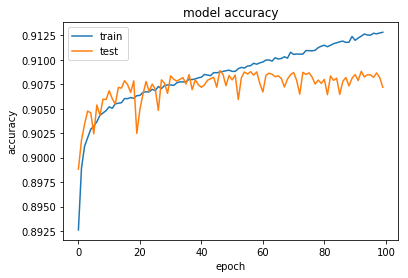
### Convolutional Neural Networks

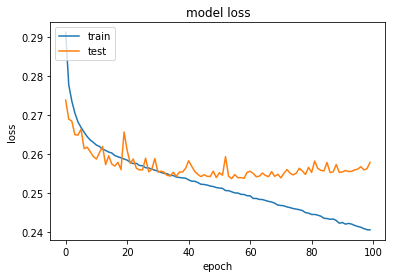
#### Stage 1



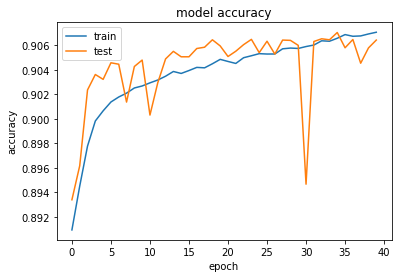


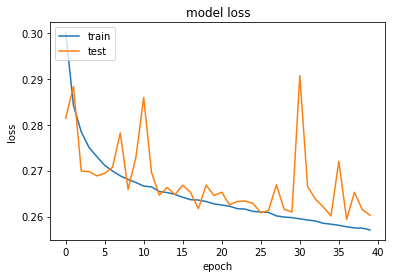
#### Stage 2





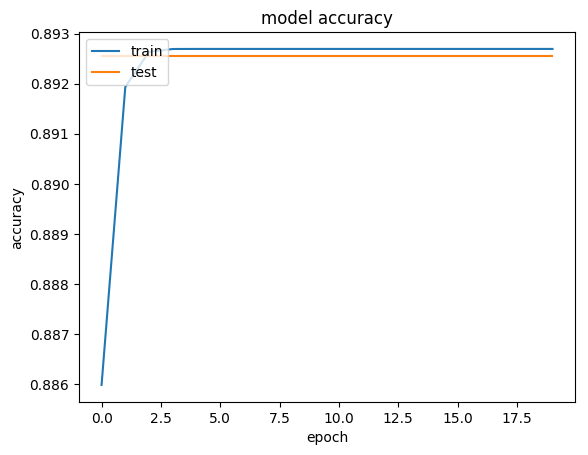
#### Stage 3

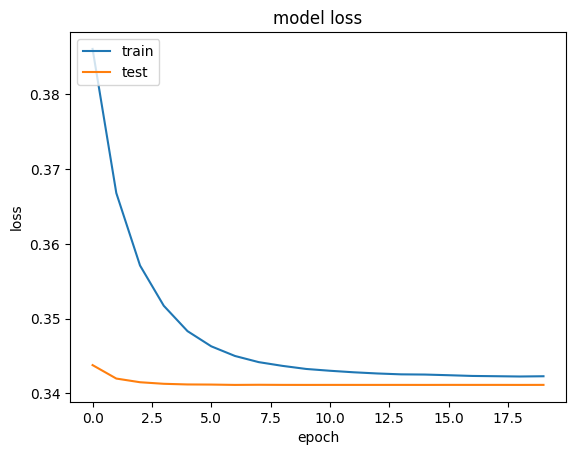




#### Stage 4

#### Stage 5





## Autoencoders



#### Boundary-Seeking Generative Adversarial Network

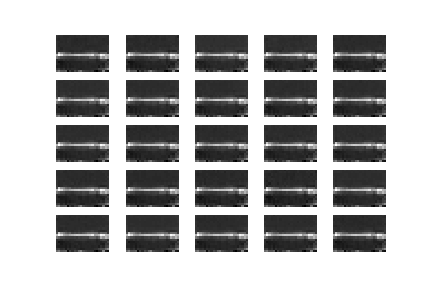
100 Latent dimensions

Adam optimizer with learning rate = 0.000001 and a batch size of 32

Generator with 8 hidden layers with 128, 256, 256, 256, 512, 512, 512 and 1024 nodes, using leaky ReLU activation and an output layer using tanh activation

Convolutional discriminator using two convolutional layers, max-pooling and 5 hidden layers with 1024, 512, 256, 128 and 64 nodes and a single node output layer with sigmoid activation.

Example output after 29000 epochs:



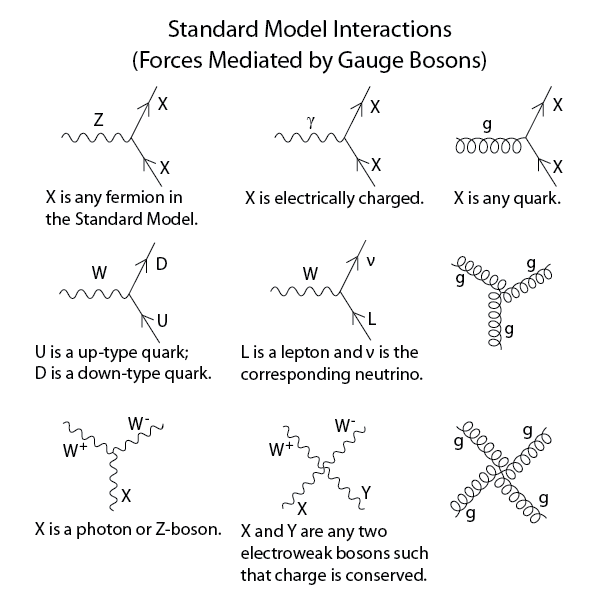
Appendix i: Standard Model Vertices

The properties of the bosons in the associated quantum field theory for the various forces of the Standard Model (i.e. QCD for the strong force, EWT (QED for the electromagnetic force and for the weak force), along with their coupling with the spin-half fermions, are illustrated by three-point interaction vertices of a gauge boson with an incoming and outgoing fermion. Each of these interactions also has an associated coupling strength [1].

A particle will only couple with the force-carrying boson if it carries the interaction’s charge, for instance quarks are the only particles that carry colour charge and are therefore the only particles that can participate in the strong interaction with a gluon; similarly, only charged particles can interact with photons; but since all 12 of the fundamental fermions listed in Table 1 carry the weak isospin charge involved in the weak interaction, they all participate in this interaction [1].

The weak charged-current interaction differs from the other forces in that it is involved in the coupling of different flavour fermions. The bosons carry charges +e and −e respectively, so in order for electric charge to be conserved, this interaction can only occur between pairs of fermions that differ by one unit of electric charge [1].

Figure 1 shows the main Standard model interaction vertices in the form of Feynman diagrams.



**Figure 1: Standard model interaction vertices [2]**

Appendix J: Data Center

CERN has a data centre with over 174,000 processor cores, 150,000 Terabytes (TB) of Disk space and over 1,000 TB of random access memory (RAM) [14]; this main datacentre is connected both to its extension in Budapest, Hungary and the multi-tier Worldwide LHC Computing Grid (WLCG), all of which operates at a data transfer rate of around 10 Gigabytes/second (GiB/s).

This is wrong:

To calculate the centre-of-mass energy at collision-time, we do:

= 13 TeV [1]

This equation is derived from the relativistic relationship between energy and momentum, where the rest energy (invariant mass of a particle) is the familiar and the kinetic energy from acceleration is . To simplify the equations, the speed of light, is set at a constant [23].

TOTEM and LHCf are smaller experiments focused on particles emitted in the forward direction during non-central collisions, TOTEM investigates particles produced during non-central collisions on either side of the CMS experiment, while LHCf does the same for non-central collisions at the ATLAS experiment [25]. LHCf uses some of these forwardly thrown particles produced at the LHC as a simulated source of cosmic rays to complement the calibration and interpretation of large-scale cosmic ray experiments [29].

MoEDAL is the most recent experiment at CERN and searches for a hypothetical magnetic monopole particle; theoretically envisioned, the magnetic monopole would be a subatomic particle with its own magnetic charge, whose evidence of existence would manifest as extensive damage to the MoEDAL detector [30].

. The ROOT forums allow users of the platform to report bugs and suggest fixes and in this way contribute to the platform without being part of the official development team

Upon installation, running the following line in a Unix terminal

> echo $ROOTSYS

will print the symbolic path to the top of the ROOT directory, e.g.

/Users/gerhard/root

Looking at the contents of this directory, $ROOTSYS/bin contains executables such as the main ROOT executable, daemons for remote ROOT file access and authentication of parallel processing capabilities, etc.

$ROOTSYS/lib contains the libraries for the C++ interpreter, image manipulation, ROOT base classes, as well as interfaces with event generators.

Additional directories exist, i.e. $ROOTSYS/tutorials which contains example .C macro files, $ROOTSYS/test which contains .cxx files and $ROOTSYS/include which contains the .h header files.

Looking at the detector geometry in more detail, we first find, closest to the collision area, a central barrel part for measuring photons, electrons, hadrons, as well as a forward muon spectrometer, all of which is embedded in a large solenoid magnet and which covers polar angles between 45°-135°. Moving outward from the first layer of the central barrel, we find an inner tracking system (ITS, Figure 10), consisting of 6 planes of silicon pixel detectors (SPD), silicon drift detectors (SDD) and silicon strip detectors (SSD), which provide for high resolution particle detection [40].

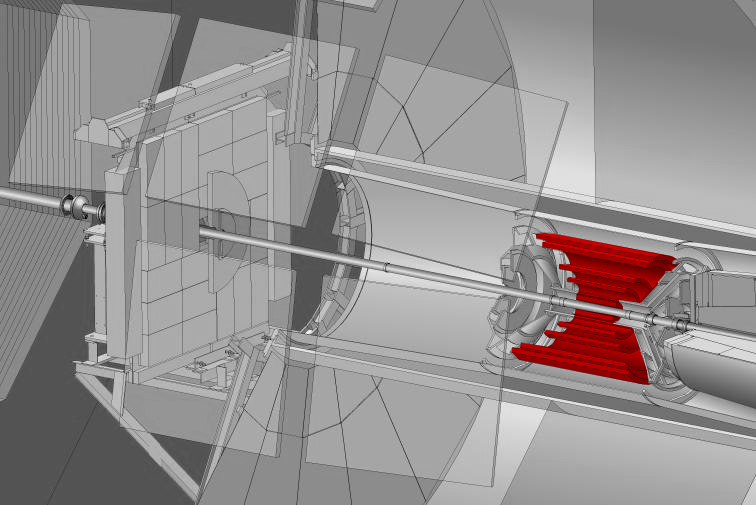


Figure 10: ALICE Inner Tracking System (SPD, SDD, SSD) [39].

The main functions of the ITS are: 1) the reconstruction of secondary vertices in the decay of strange- and heavy flavour particles, 2) particle identification and tracking of particles with low momentum, and 3) improving the resolution of impact parameters and momentum. The outer SSD detectors have analog readout for particle identification via dE/dx (see section 2.4.1 The Bethe-Bloch Curve), in the non-relativistic (i.e. low ) region.

Next, as we move outwards, we find the Time-Projection Chamber (TPC, Figure 11).

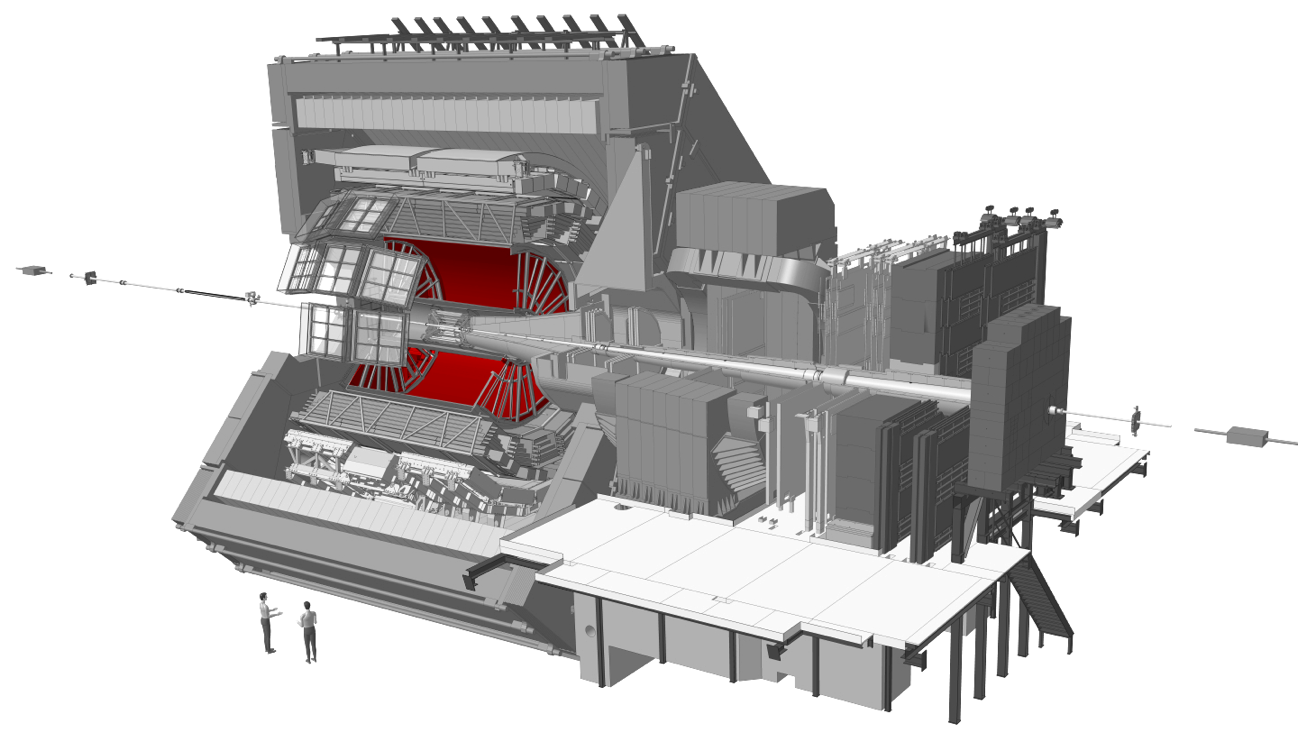


Figure 11: ALICE TPC [39].

As the main tracking detector, the TPC is a conservative system, sacrificing data volume and speed for redundant tracking mechanisms, which guarantee reliable performance, by ensuring good double-track resolution and by minimising space charge distortions [40].

After the TPC, we find three Time of Flight (TOF) particle identification arrays (Figure 12).

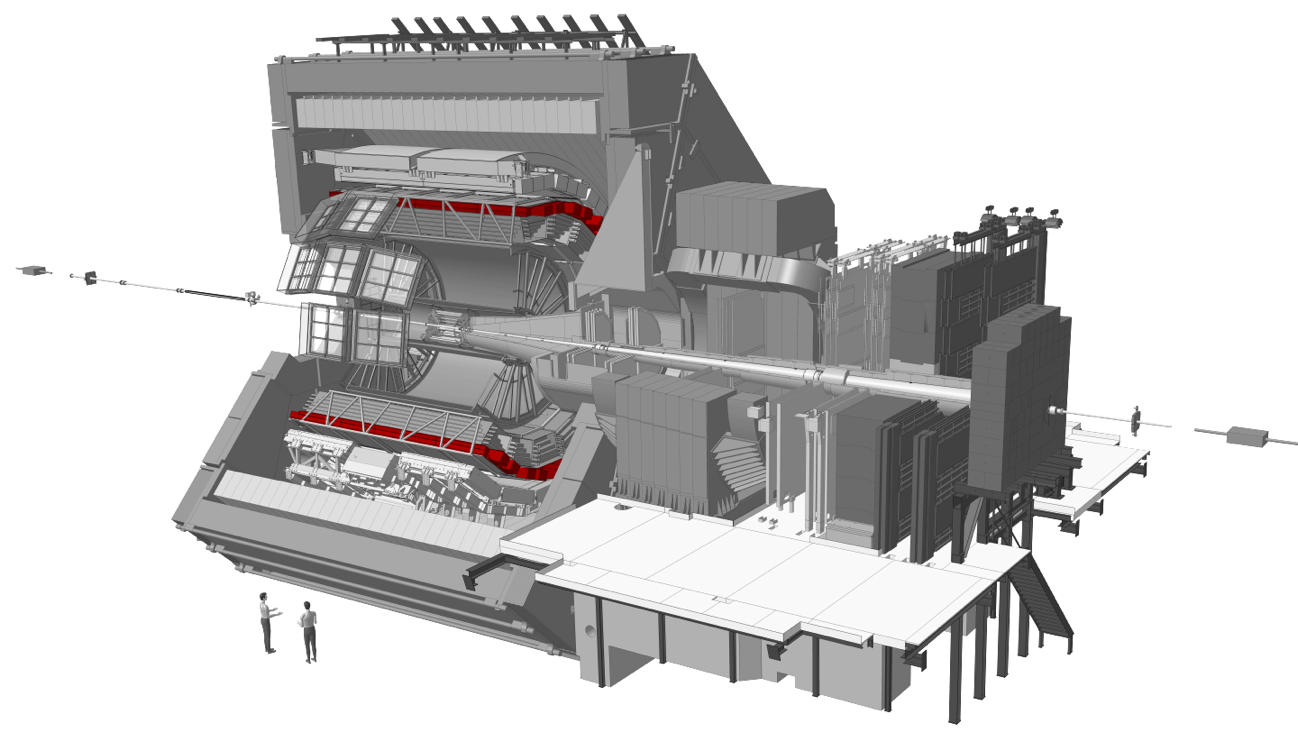


Figure 12: ALICE TOF [39].

Optimized for large acceptance and particle identification in the average momentum range, the TOF covers an area of 140 m² with 160 000 individual cells, the TOF offers time resolution of 100 ps.

Next, we find Ring Imaging Cherenkov Detectors (HMPID, Figure 13).

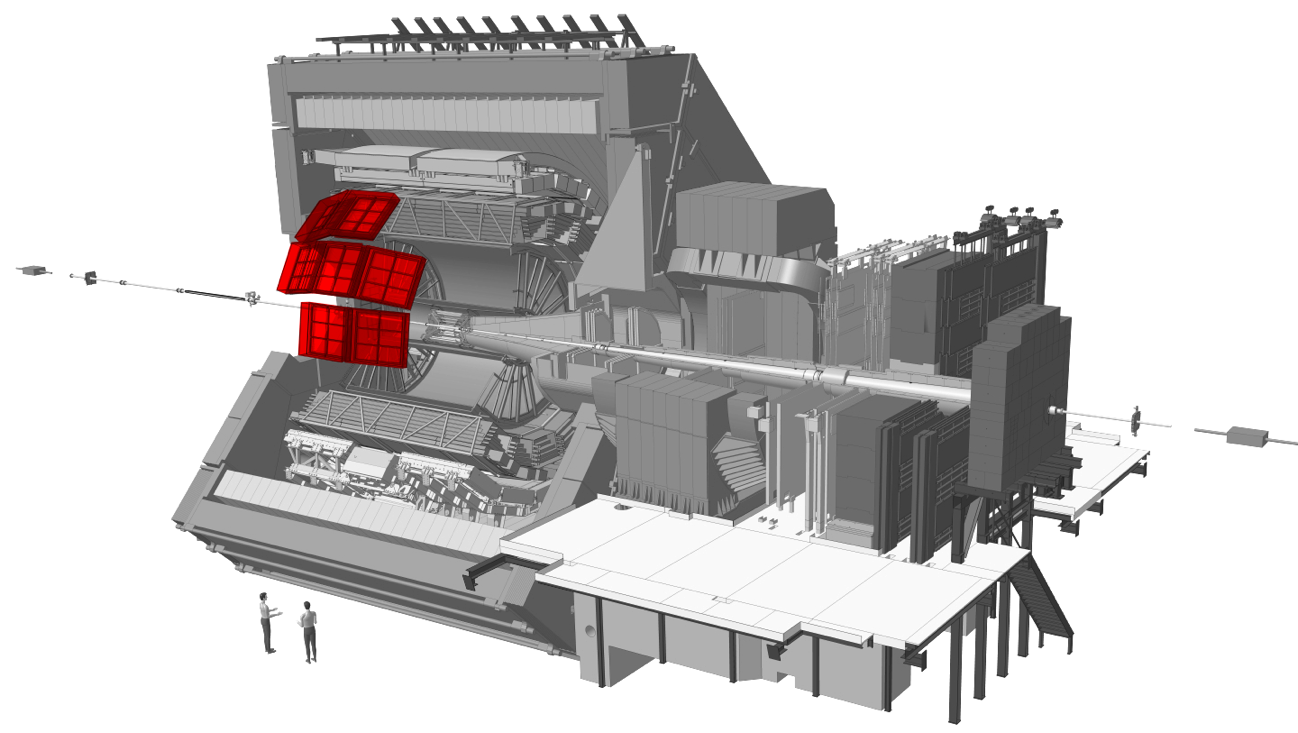


Figure 13: ALICE HMPID [39].

A single-arm detector consisting of an array of proximity focusing ring imaging Cherenkov counters, the HMPID extends particle identification (especially the identification of hadrons) towards a higher spectrum of momentum [40].

After the HMPID detectors, we get to the Transition Radiation Detector (TRD, Figure 14), part of the overarching topics of this dissertation.

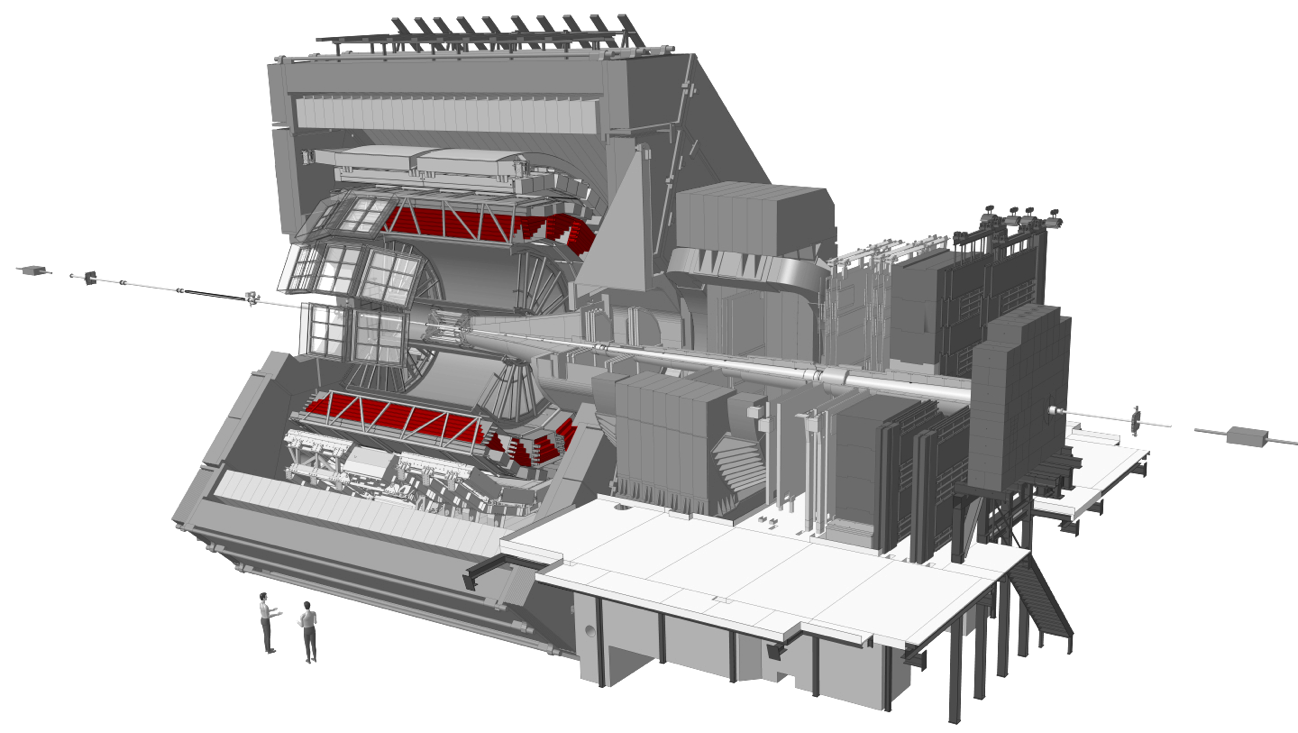


Figure 14: ALICE TRD [39]

The TRD identifies electrons of high momentum, above 1 GeV/c, to quantify production rates of quarkonia and heavy quarks in the mid rapidity (relativistic velocity) range [40]. Six time expansion wire chambers filled with are used in conjunction with attendant composite polystyrene radiators to distinguish electrons from other particles by comparing their actual energy deposition in the detector to their characteristic dE/dx curves [40].

The outer layers of the central barrel are occupied by two electromagnetic calorimeters (PHOS, Figure 15, and EMCal, Figure 16).

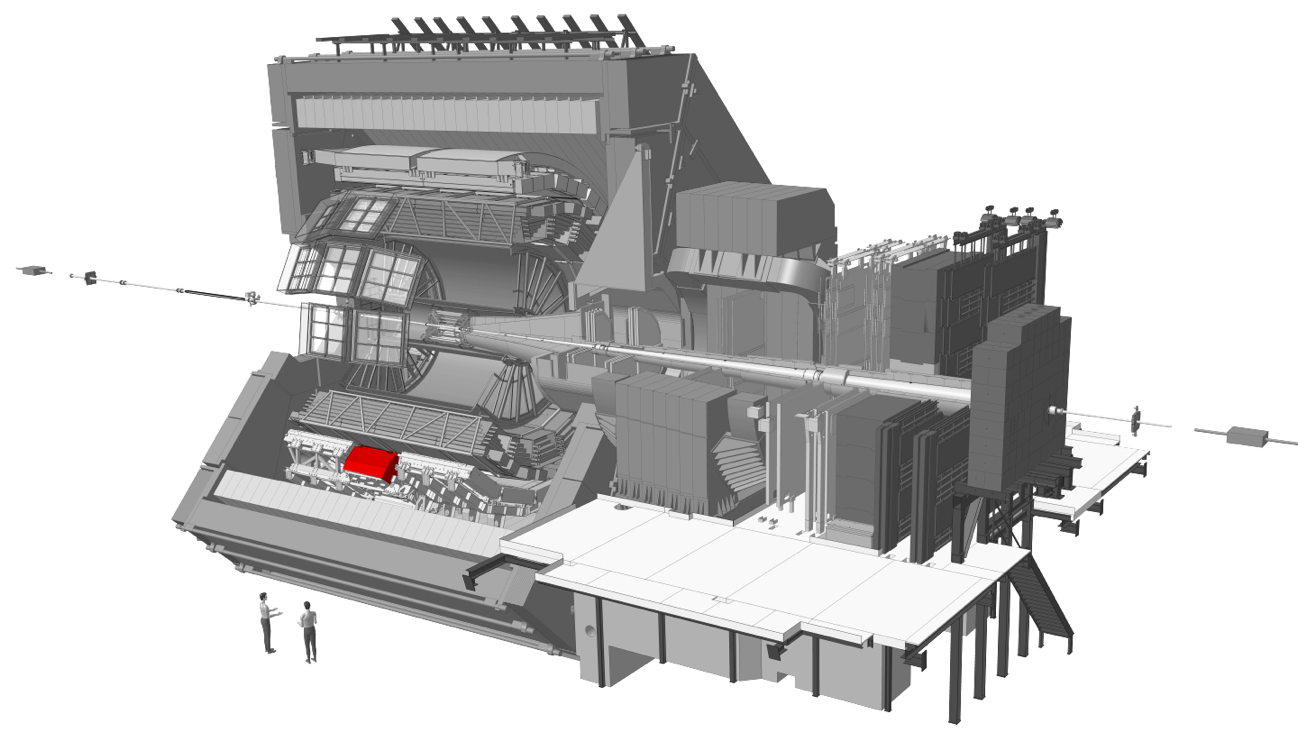


Figure 15: ALICE PHOS [39].

Another single-arm detector, PHOS is an electromagnetic calorimeter which gives a high-granularity and -resolution view of photons, to distinguish their production mechanisms (i.e. whether they arise from thermal emission or hard QCD processes). Scintillating crystals amplify the signal to give good resolution of lower energy photons. Charged particles are vetoed by a set of multiwire chambers, inwardly adjacent to PHOS [40].

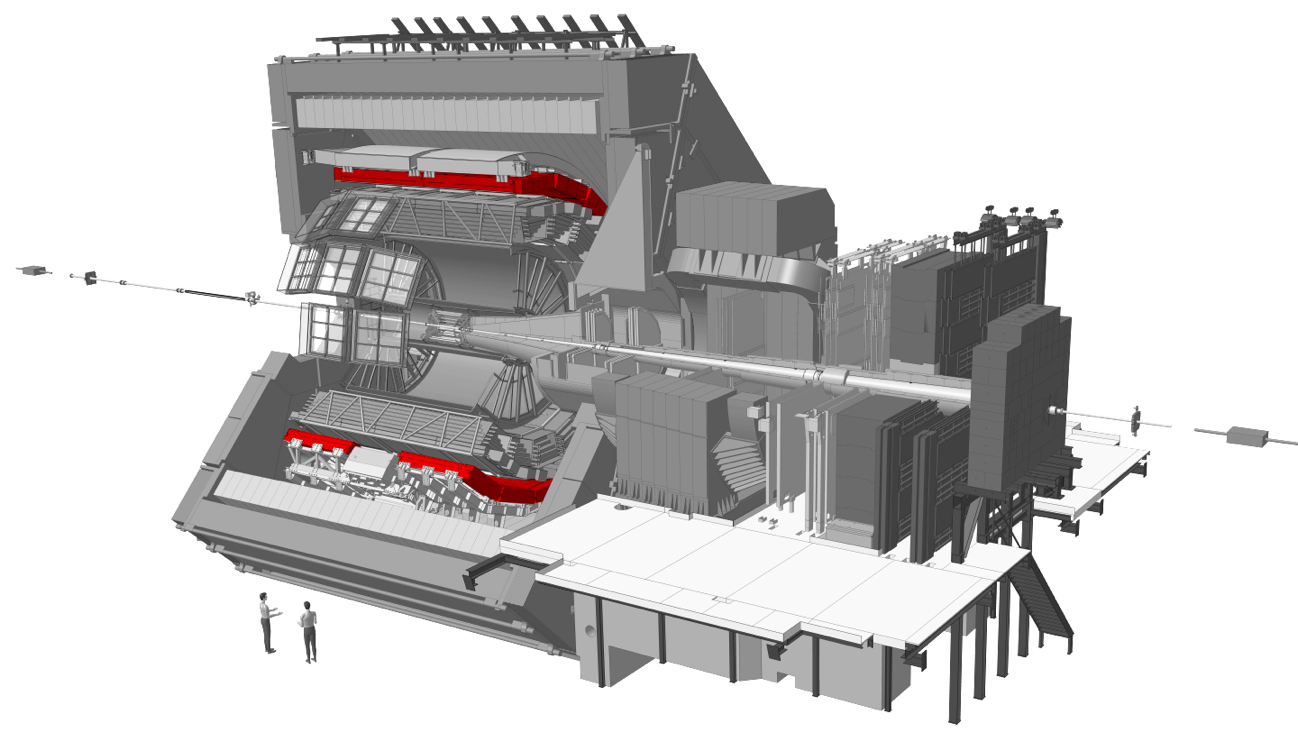


Figure 16: ALICE EMCal [39].

EMCal is a lead-scintillator sampling calorimeter, larger than PHOS, it is used in the measurement of jet production rates and fragmentation functions (functions used to calculate the probability that specific observed final states arise from a given quark or gluon) [40].

All of the detectors in the central barrel, except for HMPID, EMcal and PHOS, cover the full azimuth, i.e. they can detect particles at all angles around the central collision area [40].

Outside of the central barrel, a variety of smaller detector elements are found (V0, T0, PMD, FMD, ZDC) that are involved in the triggering of data collection for a specific event, as well as global event characterization [40].

The forward muon arm (covering angles between 2°-9° relative to the collision centre) completes the picture of the ALICE detector (Figure 17). It consists of 14 planes of triggering and tracking chambers, as well as various muon absorbers and its own dipole magnet [40].

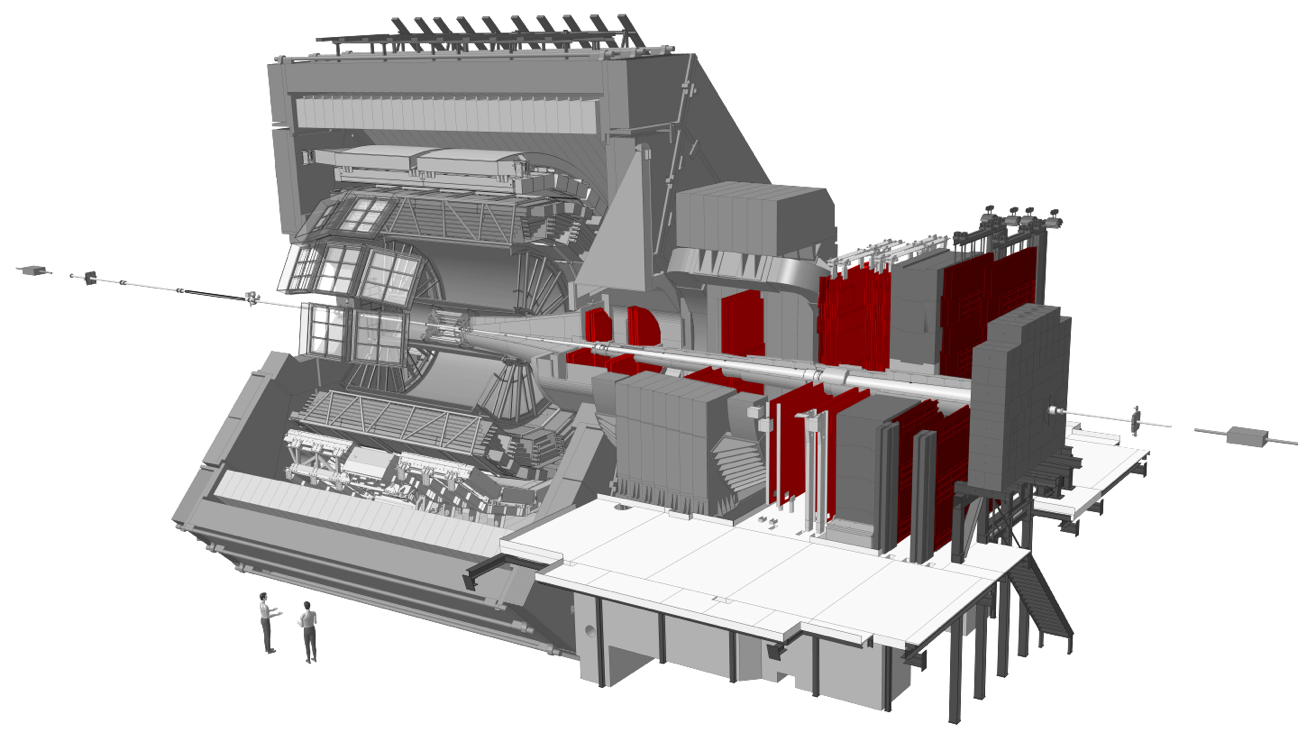


Figure 17: ALICE forward Muon arm [39].

The measurement of heavy-quark resonance production is fulfilled by the Muon spectrometer, its small angle relative to the beam-line allows acceptance down to zero transverse momentum. It is made up of a composite absorber and ten thin cathode strip planes acting as high granularity tracking stations. An additional muon filter and four Resistive Plate Chambers are employed in the processes of triggering and muon identification. The muon spectrometer is protected from secondary particles produced in the beam pipe, by a 60 cm-thick absorber tube [40].

### A Note on Geometry

Figure 18 serves as a guide to understanding the coordinate system used at the LHC and in this thesis.

The point of beam intersection (the collision centre) acts as the zero-point in geometric coordinate expressions (x = 0, y = 0, z = 0). Cylindrical coordinates are specified from this origin, with the z-axis pointing along the beam line (with positive z coordinates indicated along this plane in the direction of the muon arm) [40].

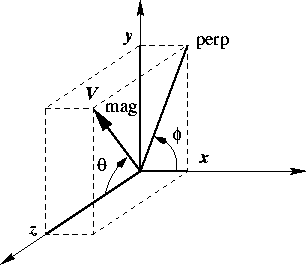


Figure 18: Cylindrical coordinates as used in geometric coordinate specifications for measurements made in experiments conducted at the LHC [42].

Where appropriate, traditional Cartesian coordinates are used, for instance when talking about the location of a detector element. In these cases, the y-axis proceeds from the origin in the direction of the wires in the Multi-Wire Proportional Chambers (MWPC, discussed in section 3.2.3) and also indicates the direction of deflection in the magnetic field, the x-axis proceeds from the origin in the direction of electron drift [40].

In order to specify the cylindrical coordinates (ρ, θ, ϕ) of a point P, one can firstly obtain ρ, by measuring the distance from the origin to point P. Next, one would project a line from P onto a point Q on the xy-plane, to obtain θ, as the angle between the positive x-axis and the line segment from the origin to point Q. Finally, one would calculate ϕ as the angle between the positive z-axis and the line segment from the origin to point P [43].

An additional geometric term used in HEP literature is pseudorapidity, , which is a specification of a particle’s angle relative to the beam (z-) axis.

## Deep Learning within the Context of Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a branch of Computer Science concerned with getting computers to perform tasks that are characteristic of those performed by the human mind. The field of AI encompasses both hard-coded rule-based programs (known as the knowledge base approach to AI, which has largely remained ineffective), as well as Machine Learning, which is an approach to AI which aims to get computers to perform these tasks without explicitly coding the solutions for them [38].

The success of Machine Learning algorithms is largely determined by the representation of the data fed through them. Often, a large amount of an AI practitioner’s time is dedicated to engineering the right feature-set to hand to a simple machine learning algorithm [38].

In the case of machine learning for image classification, which loosely ties back to some of the aims in this project, it is not always immediately obvious as to which features will be informative to an ML algorithm. For example, feeding raw pixel values into a linear regression model should not be very effective, since images vary in terms of positional information, lighting, sharpness, rotation, etc. [38]

Representation learning is a solution to feature generation in which ML is applied, not only to map from a feature set to an output, but also towards automatically learning the most useful representation of the data; usually this representation will encompass identifying the major factors of variation which effectively explain the observed data and discarding those which are not useful to the algorithm [38].

Deep Learning is an approach to representation learning which constructs useful representations based on a combination of simpler representations. In fact, the basic unit of a neural network is the perceptron, which in itself is a very simple function, but once compiled into a Multi-layer Perceptron, the rich texture of the input data distribution can be very accurately captured, since useful features discovered in the first layers of such a neural network can be combined in various ways to create additional useful features [38]. Continuing with the image classification example, an early layer of a convolutional neural network may detect edges in an image, the next layer may detect corners and shadows, and layers further down will ideally detect actual visual elements (faces, car lights, arms, etc.) [38].

## Mathematical Background for Deep Learning

### Rosenblatt’s Perceptron

The original Rosenblatt paper [39] outlining the concept of the “perceptron” aimed to develop a theory to explain: 1. How sensory information is detected by biological organisms, 2. how that information is subsequently processed and stored and 3. how mental comprehension or organismal behaviour (which he termed “preference for a particular response”) was driven by the first two processes.

He outlined a mathematical framework for these mechanisms, at the hand of the following constructs:

1. **S-points:** sensory units which can possess any of a number of response curves based on the signal strength of incoming information

2. **A-units:** association cells located in an “association area” , which in some of his models was preceded by a “projection area”

3. S-points are connected in specific ways to A-units and forward their stimulus response to them, in the form of an inhibitory or an excitatory impulse

4. : A threshold value assigned to each A-unit dictates whether it will fire, based on the algebraic sum of excitatory and inhibitory signals received, from either S-points or preceding A-units

5. The connections between S-points and A-units, and between A-units themselves is random, and not all elements of such a network are connected to each other

6. Response units, , receive a large number of inputs from the set, called its source-set, and have feedback mechanisms to A-units in its source set. [39]

He put forth various models for response curve summation and how these networks would learn [39], but while the mathematical constructs he proposed were oversimplifications of the complexity of biological brains, they were found to be extremely useful in training computers to emulate their capabilities.

For k = hidden layers, , we compute the element-wise gradient on the layer’s output (before the non-linear activation function is applied):

And the gradients on the weights and the bias term:

Here, represents the weight decay penalty, where the size of the weights are constrained, in a manner inversely proportional to . A regularizer is added to the loss, where contains all the weight and bias parameters.

This gradient is then propagated to the activations of the preceding layer:

Regularization in deep learning models often involves limiting the capacity (the hypothesis space) of an ANN by introducing a parameter norm penalty to the loss function J. The loss function regularized in this fashion is denoted by J̃, as follows:

Where α is a weighting hyperparameter, determining the extent of contribution of the parameter norm penalty to the magnitude of the regularized loss function J̃, i.e. setting α = 0 eliminates regularization and increasing its value results in more regularization [38].

Various norms Ω can be used in such a setup and can be applied to the entire set of network parameters θ or a specific subset, e.g. all the weights can be regularized, but all the bias terms can be set to escape regularization, because weights encode the interaction between two variables under a variety of circumstances, whereas bias terms only affect the output of one variable [38].

Ideally, each ANN layer should have its own α coefficient, but doing so increases the search space for the optimal value, so a global α is sometimes used in practice [38].

##### A Note on Norms

Norms are a means of measuring the size of a vector, by mapping them to non-negative values, by satisfying the following properties:

In general, the norm is specified by:

##### Weight Decay Regularization

The parameter norm is a simple regularization strategy which shrinks the weights of an ANN closer to the origin by adding the squared and weighted parameter norm penalty

to the objective function [38].

The norm is known as the Euclidean norm, because it gives the magnitude of the Euclidean distance from the origin to the point defined by . It is squared in this regularization technique for computational efficiency, because calculating the derivative with respect to each component of the unsquared norm involves all its elements, whereas the derivative for each component of the squared norm depends only on the corresponding element of [38].

Figure 17 (RHS) illustrates the manner in which introducing an norm penalty introduces an additional constraint on the objective function, i.e. having to minimize the magnitude of the norm in addition to minimizing the loss function causes the weights to be shrunk, since this larger regularized loss function is interpreted as having higher variance [38].

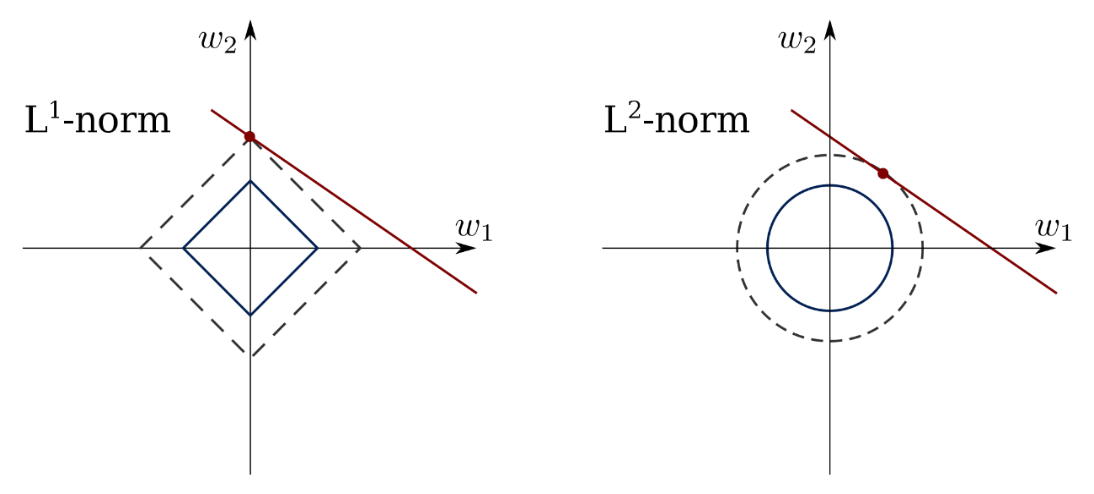


Figure 17: and norm penalties

##### Regularization

regularization adds a slightly different weighted parameter norm penalty

to the objective function.

When regularization is used, as the sum of the absolute values of the weights of the ANN increases, the loss function will also increase, as it does for regularization; but in contrast to regularization, allows weights to be shrunk down to zero, resulting in a more sparse neural network, depending on the magnitude of the weighting parameter . This phenomenon allows for better feature selection, by reducing the amount of connections in the network and therefore removing the influence of some features on its output [38].

When using either or regularization, care has to be taken to select the right level for , since a large could result in the backpropagation algorithm getting trapped in a local minimum or where the weights are shrunk by so much that they can’t impart any useful information to the next layer [38].

##### Ensembled Models

Bagging and other model averaging techniques involve training a multitude of models and allowing each of them to vote towards the outcome, making use of the principle that a number of Deep Learning models which have each been set up differently should not all make the same “cognitive errors” when learning useful representations to inform accurate predictions on the test set [38].

Bagging, in particular, requires construction of multiple training datasets by sampling with replacement from the full training dataset, resulting in around a third of the full training observations not being present in each of the resampled training sets, and different observations being missing in each [38].

Since random weight initialization and random minibatch selection can result in slightly different weight parameterisation, even when the same architecture is trained multiple times on the same dataset, model averaging is a highly reliable way to reduce overfitting [38].

Boosting is an alternative approach to ensembled methods, which actually increases the capacity of the ensemble by learning based on the variance of previous neural networks by adding additional neural networks sequentially, or even by incrementally introducing hidden units to a single ANN [38].

##### Early Stopping

By saving the parameter setting at the conclusion of each epoch during training, one can return the network to the parameter setting where the validation error was at its lowest (the point at which the network started overfitting to the training set) [38].

One can also prevent a model from passing that point by specifying early stopping criteria, which will kick the neural network out of training when a defined minimum improvement on the validation error has not occurred for a defined number of epochs [38].

Computational efficiency is maintained by checking the abovementioned conditions at specified training intervals, i.e. not checking whether early stopping criteria have been met after each epoch. Storing parameter settings can be made more efficient by saving in a slower form of memory, such as hard disk space to keep available random-access memory or GPU memory space sufficient for model training [38].

Once early stopping has been reached, the checkpointed model can be trained further by adding the previously held out validation data to the training data and monitoring the objective function as a guide for when to interrupt training [38].

Alternatively, once early stopping criteria are reached, one can retrain a completely new neural network, with the same hyperparameters as the stopped network, for the number of epochs it ran, but this time using the full training + validation data for training [38].

Early stopping is often used in conjunction with other regularization techniques, since it is unobtrusive towards the learning dynamics, i.e. it does not change how the neural network arrives at its optimal weights, it simply changes when to stop adjusting them to prevent overfitting [38].

Full convolutions result from applying enough zero-padding to allow each pixel to be visited k times in each direction of the convolution operation, and therefore should result in an output with m+k-1 pixels. This results in output pixels near the border being influenced by fewer pixels than output pixels near the centre, making the kernel harder to train [38].

The ideal amount of padding generally lies between the amount of padding required to achieve valid- and same convolutions [38].

## Recurrent Neural Networks

Recurrent neural networks (RNNs) are specifically designed to process sequential values. Parameter sharing is an essential aspect of RNNs and facilitate the detection of patterns that could potentially occur in more than one place in the sequence; this family of ANNs also accounts for sequences of differing length [38].

Parameter sharing in RNNs manifest in the form that each element of the output is a function of previous elements of the output within a specified range and is updated using the same rule used to update previous elements of the output [38].

The input vector to an RNN will be vectors with timestep t consisting of a range from 1, …, [38].

Recurrent neural networks extend the concept of a computational graph to include cyclical connections, where the present value of a variable is understood to have an influence on its future value [38].

### Computational Graphs

Computational graphs are visual depictions which formalize a set of operations applied to an input vector, for example the computational graph formalizing the ReLU activated output of a hidden unit, i.e. , would look as follows:

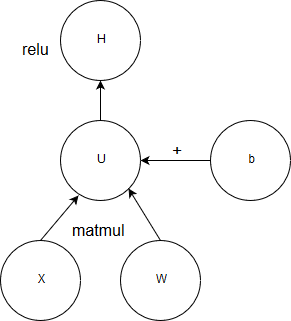


Figure 20: ReLU activated hidden unit in a Neural Network depicted as a computational graph

In RNNs, computational graphs manifest as repetitive chains of operations which result in parameter sharing across neural network architectures [38].

As an example, a dynamical system is classically expressed as:

Here, the state of the system at time t, , explicitly depends on its state at the previous time step (t-1).

This graph can be unfolded for a finite number of timesteps, , by applying the above expression times, e.g. if =3:

The above equation can be represented as an acyclic graph, which does not make use of recurrence, as follows:

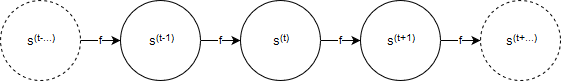


Figure 21: Acyclic computational graph of a dynamical system

If we extend this to express the dynamical system’s state at any point being informed by all the previous states of the system, the equation becomes:

This is the basic formula upon which RNNs are built, where the “states” of the system are the neural network’s hidden units, i.e.

[38].

### Long Short-Term Memory

Long Short-Term Memory (LSTM) recurrent neural networks are a highly successful class of RNN which deals with the problem of exploding or vanishing gradients introduced by other RNN implementations by enforcing constant error flow through the internal states of special units, called memory cells, shown in a computational graph in Figure 22 [42].

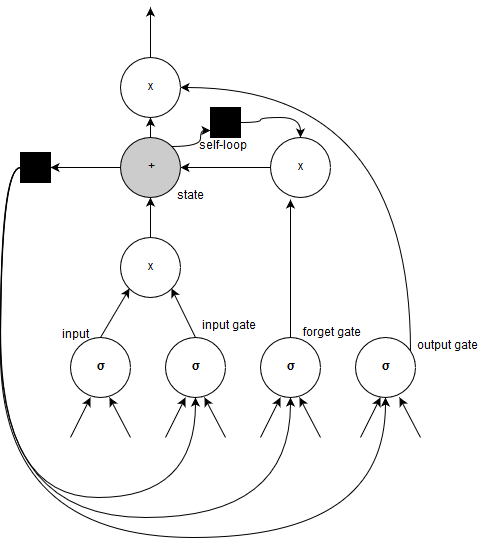


Figure 22: Graph-based representation of special LSTM units

Multiplicative input and output gates protect other units from irrelevant inputs and currently irrelevant stored memory states, respectively. Each memory cell as shown above consists of a central linear unit with a self-connection which is fixed [42]. In this way, a memory cell can decide whether or not to save information about its current state, based on inputs from other memory cells.