



Laboratory of Computational Physics - Module A

Advanced DQM for a Drift Tube Detector

Group 13:

- Pietro Cappelli
- Alberto Coppi
- Giacomo Franceschetto
- Nicolò Lai



Project outline

Starting from data acquired by a cosmic muon detector of drift tubes at Legnaro laboratories we focused on:

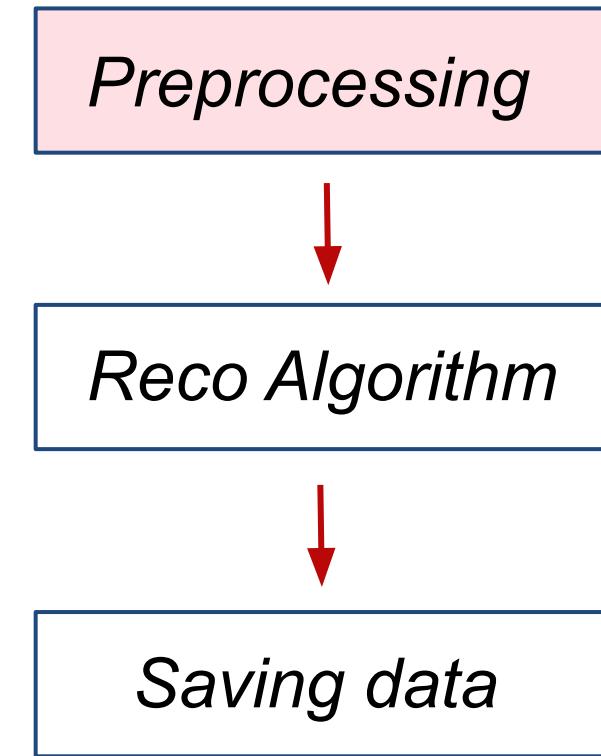
- **building an output dataset** with **drift time** and **crossing angle** of the muon track as features
- **optimizing the reconstruction procedure** to obtain a more efficient and reproducible routine
- **studying the correlation** between these two features
- **testing** the performance of **NPLM** using the output dataset

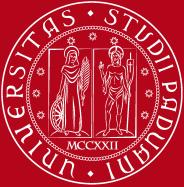


Building the dataset

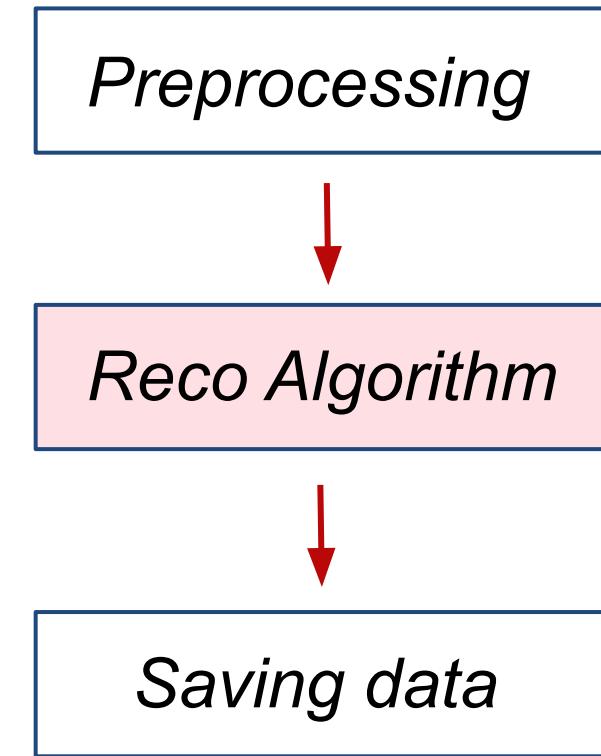


Building the dataset *outline*



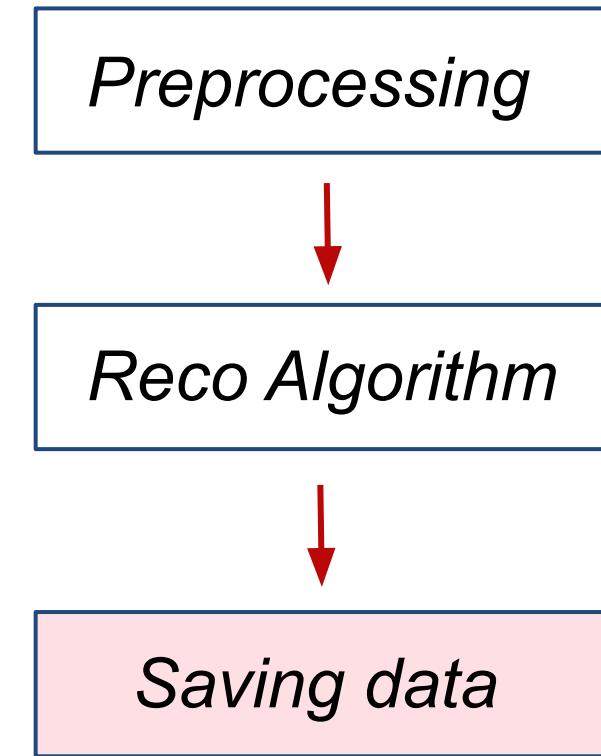


Building the dataset *outline*





Building the dataset *outline*





Preprocessing *the path from raw to readable data*

HEAD	FPGATDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	2	0	35	1379482729
1	2	0	56	1379482738
2	2	1	122	1379482739
3	2	0	98	1379482743
4	2	0	101	1379482743



	FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB
0	0	100	1379484750	0	4	63.0	19.5	-5.07	63.27
1	1	35	1379484750	2	1	42.0	-19.5	43.60	39.65
2	0	101	1379484750	0	2	63.0	-6.5	45.77	60.54
3	1	81	1379484750	3	2	-147.0	-6.5	107.27	-152.78
4	0	38	1379484750	1	3	84.0	6.5	96.43	78.81



Preprocessing *the path from raw to readable data*

HEAD	FPGATDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	2	0	35	1379482729
1	2	0	56	1379482738
2	2	1	122	1379482739
3	2	0	98	1379482743
4	2	0	101	1379482743



	FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB	
0	0	100	1379484750	0	4	63.0	19.5	-5.07	63.27	62.73
1	1	35	1379484750	2	1	42.0	-19.5	43.60	39.65	44.35
2	0	101	1379484750	0	2	63.0	-6.5	45.77	60.54	65.46
3	1	81	1379484750	3	2	-147.0	-6.5	107.27	-152.78	-141.22
4	0	38	1379484750	1	3	84.0	6.5	96.43	78.81	89.19



Preprocessing *the path from raw to readable data*

HEAD	FPGATDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	2	0	35	1379482729
1	2	0	56	1379482738
2	2	1	122	1379482739
3	2	0	98	1379482743
4	2	0	101	1379482743



FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB
0	0	100	1379484750	0	4	63.0	19.5	-5.07
1	1	35	1379484750	2	1	42.0	-19.5	43.60
2	0	101	1379484750	0	2	63.0	-6.5	45.77
3	1	81	1379484750	3	2	-147.0	-6.5	107.27
4	0	38	1379484750	1	3	84.0	6.5	96.43



Preprocessing *the path from raw to readable data*

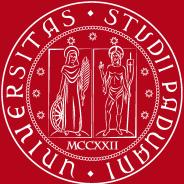
FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB
0	0	100	1379484750	0	4	63.0	19.5	-5.07
1	1	35	1379484750	2	1	42.0	-19.5	43.60
2	0	101	1379484750	0	2	63.0	-6.5	45.77
3	1	81	1379484750	3	2	-147.0	-6.5	107.27
4	0	38	1379484750	1	3	84.0	6.5	96.43
							78.81	89.19

```
# SL column
# sl -> 0: fpga 0 tdc_ch in [64, 127]; sl -> 1: fpga 0 tdc_ch in [0, 63]
# sl -> 2: fpga 1 tdc_ch in [0, 63];   sl -> 3: fpga 1 tdc_ch in [64, 127]
df["SL"] = (df["TDC_CHANNEL"] + 128*df["FPGA"]) // 64
df["SL"][df["SL"] < 2] = [int(not x) for x in df["SL"]]

# LAYER column
df.loc[(df["TDC_CHANNEL"] % 4 == 0), "LAYER"] = 4
df.loc[(df["TDC_CHANNEL"] % 4 == 2), "LAYER"] = 3
df.loc[(df["TDC_CHANNEL"] % 4 == 1), "LAYER"] = 2
df.loc[(df["TDC_CHANNEL"] % 4 == 3), "LAYER"] = 1

# assign the wire position
for layer in [1, 2, 3, 4]:
    # local wire x position
    df.loc[df["LAYER"] == layer, "WIRE_X_LOC"] = (df["TDC_CHANNEL"] % 64 // 4) * XCELL + X_POS_SHIFT[layer]

    # local wire z position
    df.loc[df["LAYER"] == layer, "WIRE_Z_LOC"] = Z_POS_SHIFT[layer]
```



Preprocessing *the path from raw to readable data*

FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB		
0	0	100	1379484750	0	4	63.0	19.5	-5.07	63.27	62.73
1	1	35	1379484750	2	1	42.0	-19.5	43.60	39.65	44.35
2	0	101	1379484750	0	2	63.0	-6.5	45.77	60.54	65.46
3	1	81	1379484750	3	2	-147.0	-6.5	107.27	-152.78	-141.22
4	0	38	1379484750	1	3	84.0	6.5	96.43	78.81	89.19

```
# HIT_DRIFT_TIME column
df["HIT_DRIFT_TIME"] = (df["BX_COUNTER"] + df["TDC_MEAS"]/30)*DURATION_BX - df["T0_NS"]

# Left, right hit positions
df["X_LEFT_GLOB"] = df["WIRE_X_GLOB"] - df["HIT_DRIFT_TIME"]*VDRIFT
df["X_RIGHT_GLOB"] = df["WIRE_X_GLOB"] + df["HIT_DRIFT_TIME"]*VDRIFT
```



Preprocessing *the path from raw to readable data*

FPGATDC_CHANNEL	ORBIT_CNT	SL	LAYER	WIRE_X_LOC	WIRE_Z_LOC	HIT_DRIFT_TIME	X_LEFT_GLOB	X_RIGHT_GLOB
0	0	100	1379484750	0	4	63.0	19.5	-5.07
1	1	35	1379484750	2	1	42.0	-19.5	43.60
2	0	101	1379484750	0	2	63.0	-6.5	45.77
3	1	81	1379484750	3	2	-147.0	-6.5	107.27
4	0	38	1379484750	1	3	84.0	6.5	96.43
							78.81	89.19

```
# HIT_DRIFT_TIME column
df["HIT_DRIFT_TIME"] = (df["BX_COUNTER"] + df["TDC_MEAS"]/30)*DURATION_BX - df["T0_NS"]

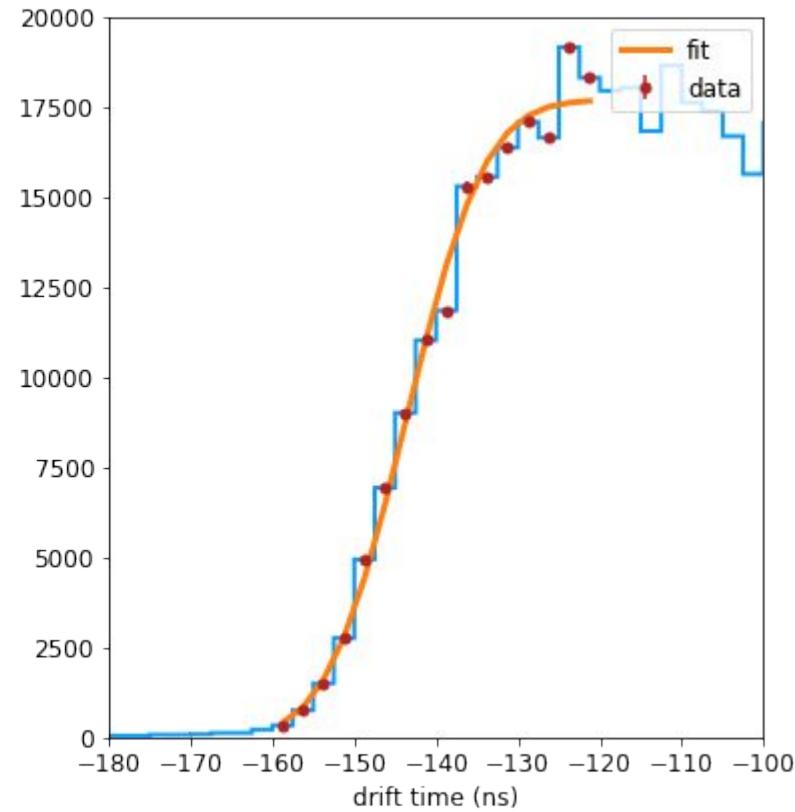
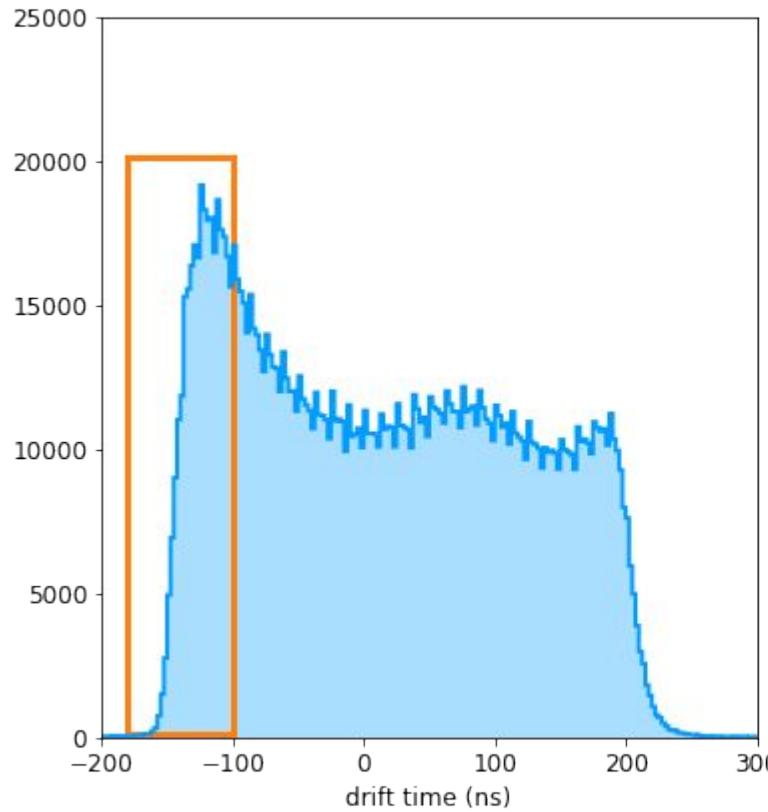
# Left, right hit positions
df["X_LEFT_GLOB"] = df["WIRE_X_GLOB"] - df["HIT_DRIFT_TIME"]*VDRIFT
df["X_RIGHT_GLOB"] = df["WIRE_X_GLOB"] + df["HIT_DRIFT_TIME"]*VDRIFT
```

$$T_0 = T_{\text{scint}} + T_{\text{scint_offset}}$$



Preprocessing *the path from raw to readable data*

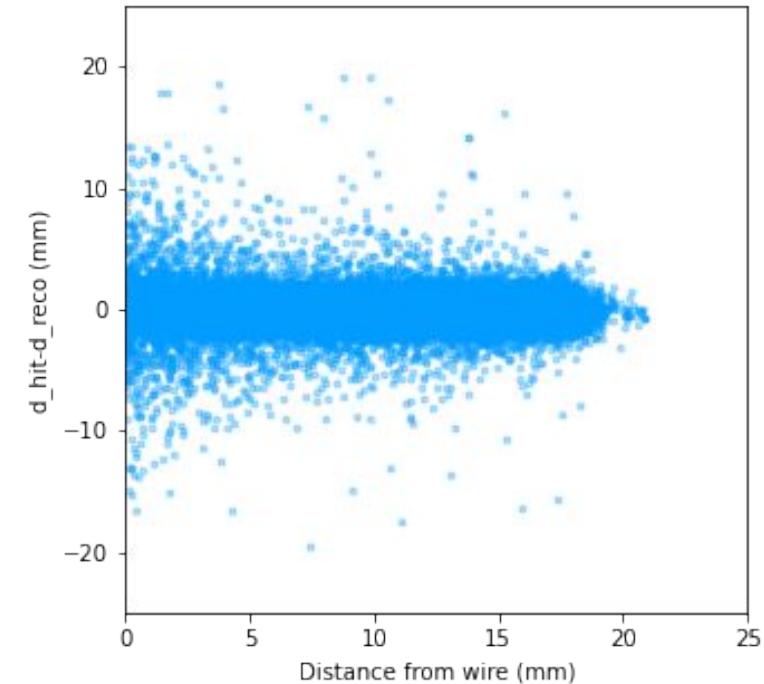
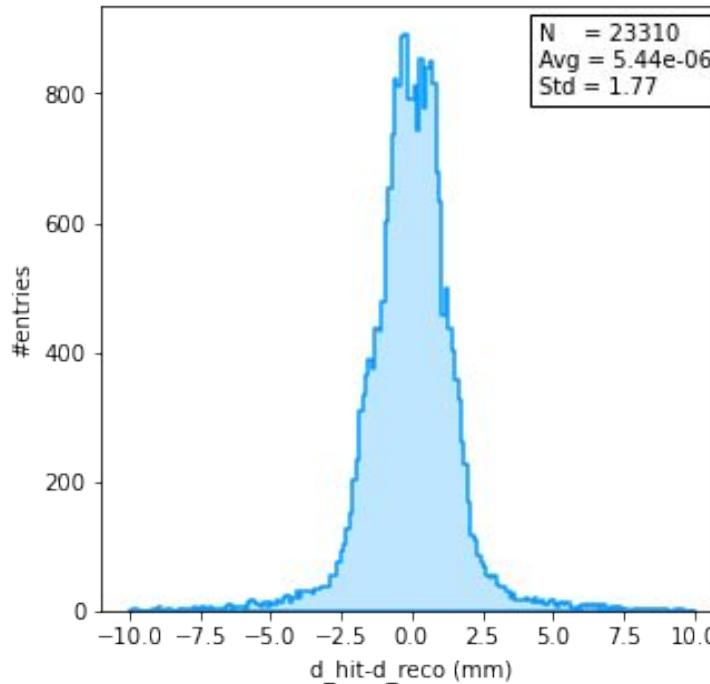
$$f(t) = \frac{1}{2}I \left[1 + \text{erf}\left(\frac{t - \bar{t}}{\sqrt{2}\sigma}\right) \right] \rightarrow T_{\text{scint_offset}} = \bar{t} - k \cdot \sigma$$





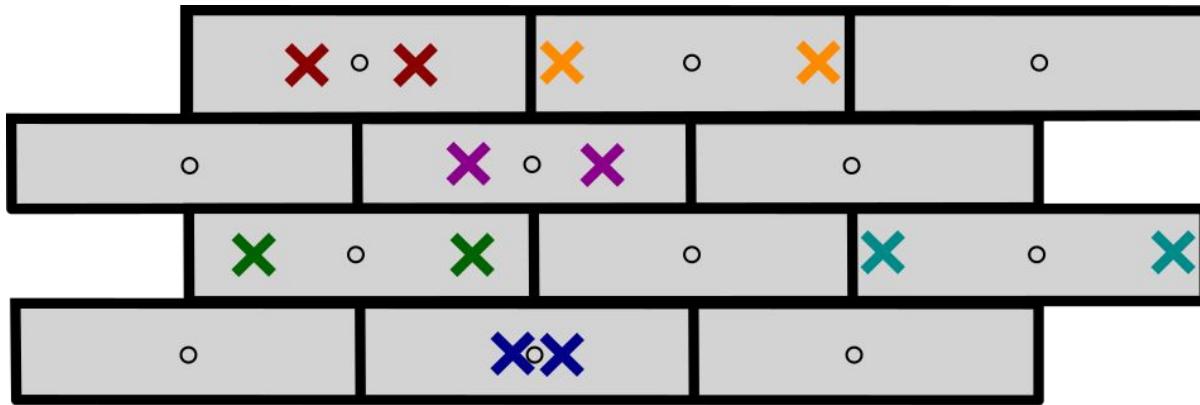
Preprocessing *the path from raw to readable data*

$$f(t) = \frac{1}{2}I\left[1 + \text{erf}\left(\frac{t - \bar{t}}{\sqrt{2}\sigma}\right)\right] \rightarrow T_{\text{scint_offset}} = \bar{t} - [k] \cdot \sigma$$



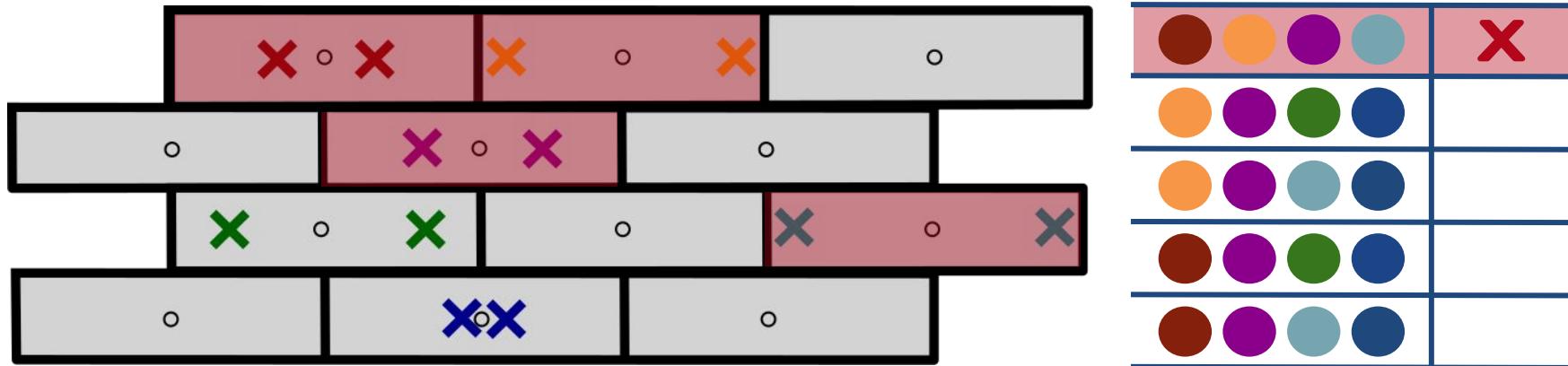


Reco Algorithm *first combinatorial step*



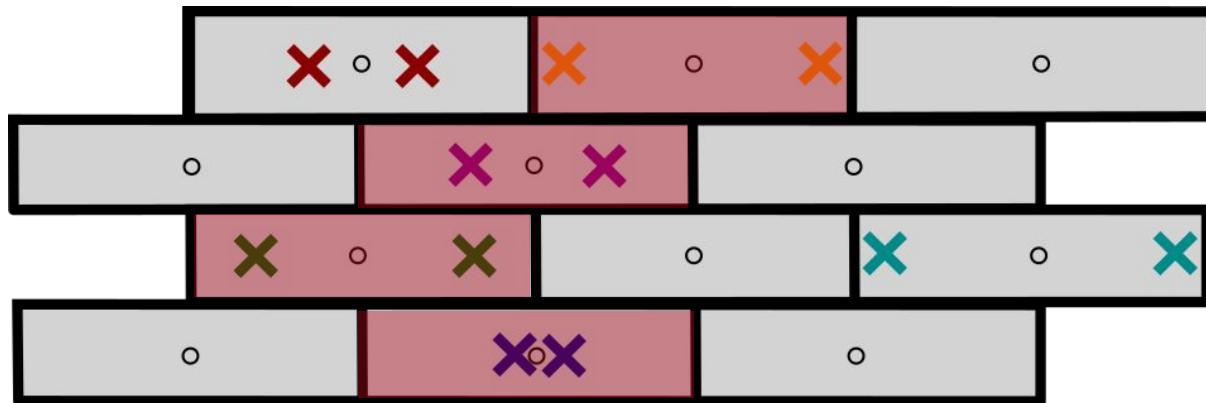


Reco Algorithm *first combinatorial step*





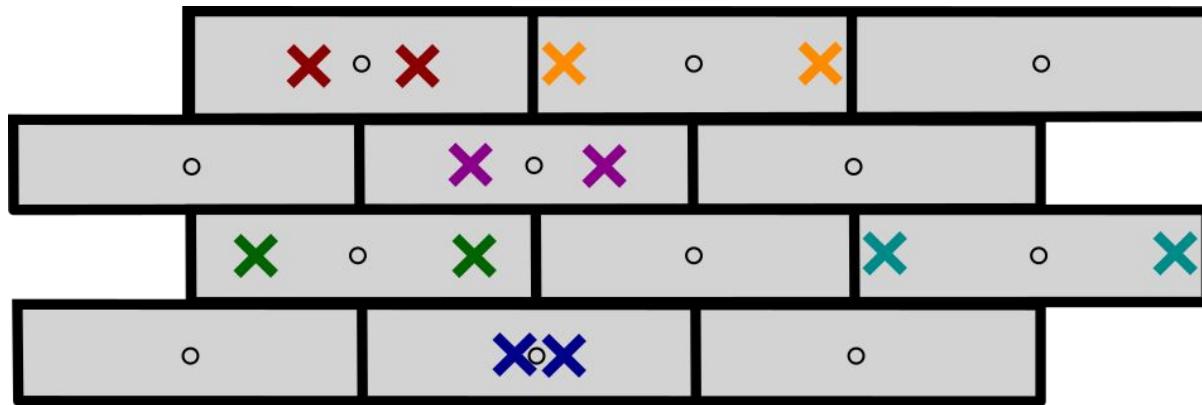
Reco Algorithm *first combinatorial step*



	X
	✓

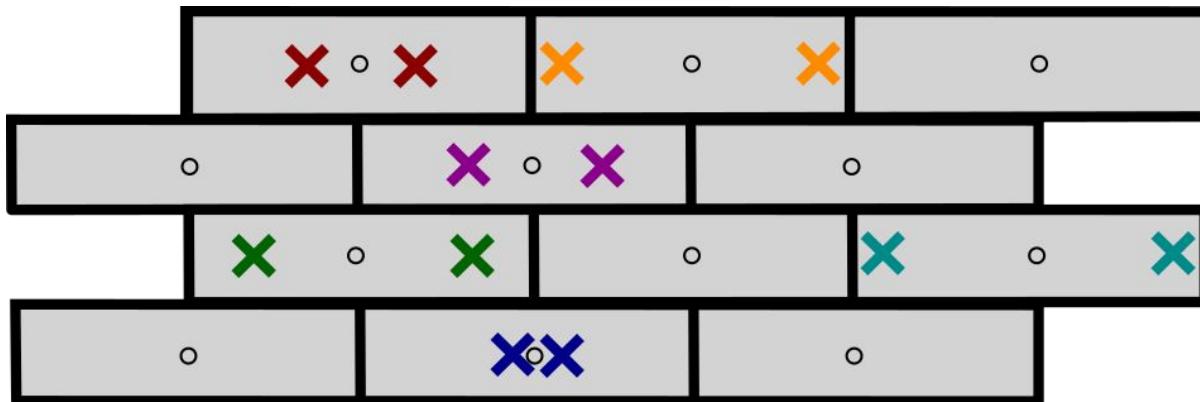


Reco Algorithm *first combinatorial step*





Reco Algorithm *first combinatorial step*

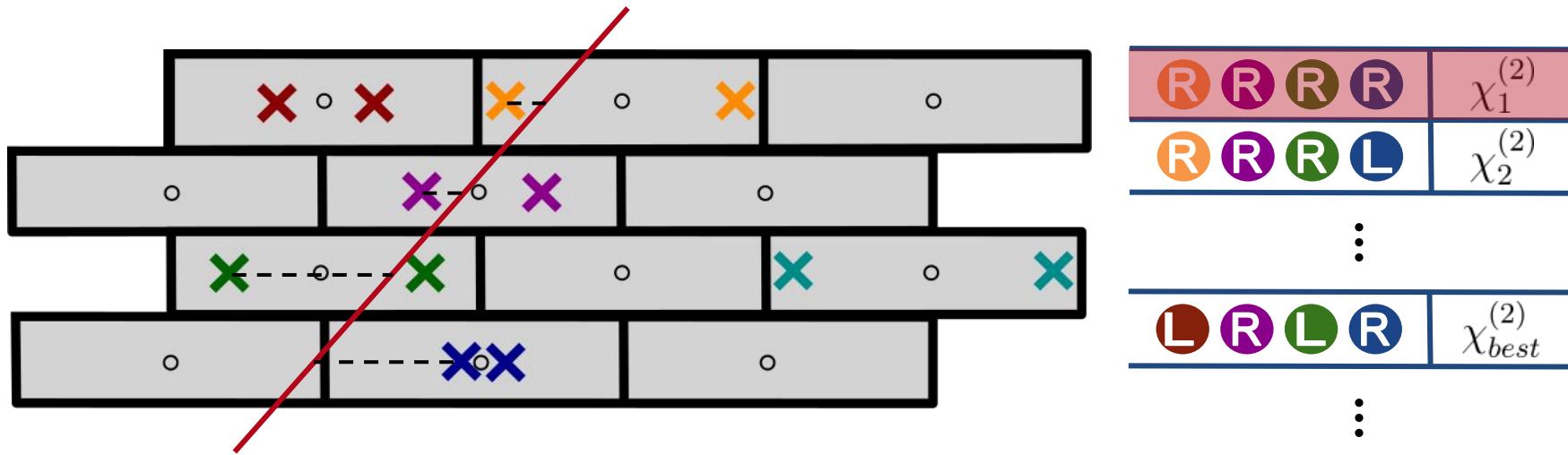


	X
	✓
	✓
	✓
	✓

```
comb = []
if len(df.LAYER.unique()) == 3:
    comb.append(df)
    tot_Hits = 3
else:
    for index in list(combinations(df.index, 4)):
        if len(df.loc[index, :].LAYER.unique()) == 4:
            comb.append(df.loc[index, :])
    tot_Hits = 4
```

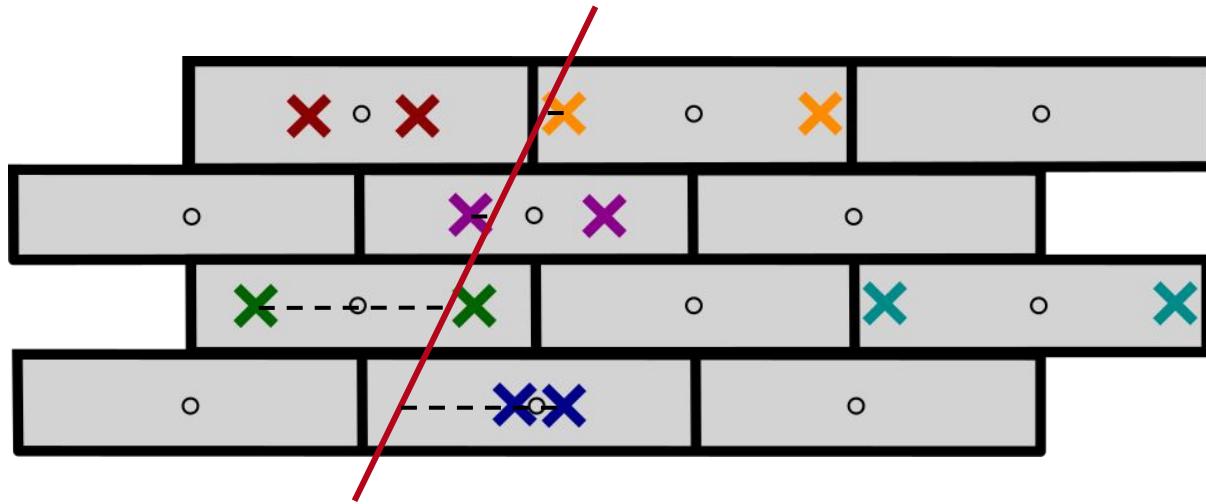


Reco Algorithm *second combinatorial step*





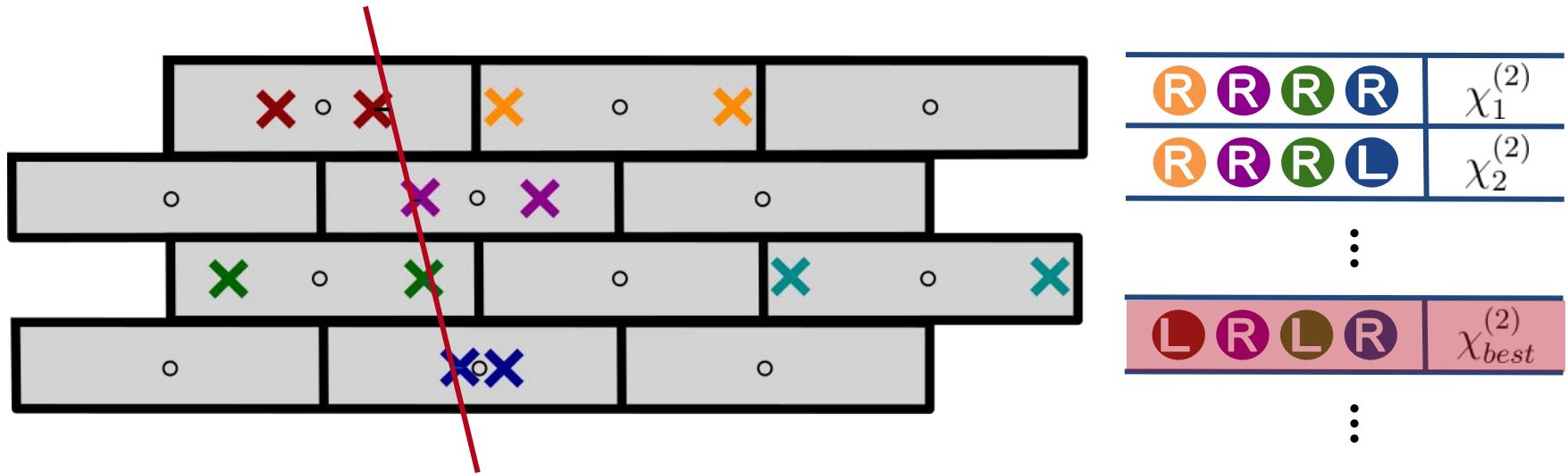
Reco Algorithm *second combinatorial step*



R	R	R	R	$\chi_1^{(2)}$
R	R	R	L	$\chi_2^{(2)}$
⋮				
L	R	L	R	$\chi_{best}^{(2)}$
⋮				



Reco Algorithm *second combinatorial step*



$$\min_{i \in \{2^{\text{nd}} \text{ step combinations}\}} \frac{|\chi_i^{(2)} - d.o.f.|}{\sqrt{2d.o.f.}}$$



best combination



Reco Algorithm *second combinatorial step*

```
min_lambda = np.finfo(float).max

for data in comb:
    X = np.array(pd.concat([data["X_RIGHT_GLOB"], data["X_LEFT_GLOB"]]))
    Y = np.array(pd.concat([data["WIRE_Z_GLOB"], data["WIRE_Z_GLOB"]]))
    for indexes_comb in list(combinations(range(len(X)), tot_Hits)):
        indexes_comb = list(indexes_comb)
        if len(np.unique(Y[indexes_comb])) == tot_Hits:
            regr_dict = linear_reg(X[indexes_comb], Y[indexes_comb])
            if abs(regr_dict["chisq_comp"]) < min_lambda:
                min_lambda = abs(regr_dict["chisq_comp"])
                xdata = X[indexes_comb]
                res_dict = regr_dict
                best_comb = indexes_comb
                best_data = data
```



Output dataset

SL	CH	HIT_DRIFT_TIME	D_WIRE_HIT	m	THETA
0	0	100	-5.066667	0.272821	-0.127657
1	0	100	149.100000	8.028462	-0.171659
2	0	100	79.933333	-4.304103	-1.141815
3	0	100	32.433333	-1.746410	-0.144280
4	0	100	39.933333	2.150256	0.209862
					11.852203

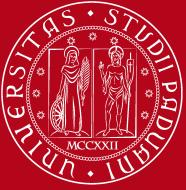
```
def saveChannels(df, OUTPUT_PATH, RUNNUMBER):

    FILE_NAME = f"RUN00{RUNNUMBER}_channels.h5"
    save_to = OUTPUT_PATH + FILE_NAME

    print("Saving data...")

    for sl in np.unique(df["SL"]):
        for channel in np.unique(df[df["SL"] == sl]["CH"]):
            df[(df["SL"] == sl) & (df["CH"] == channel)].to_hdf(save_to, key=f"sl{sl}/ch{channel}", mode="a")

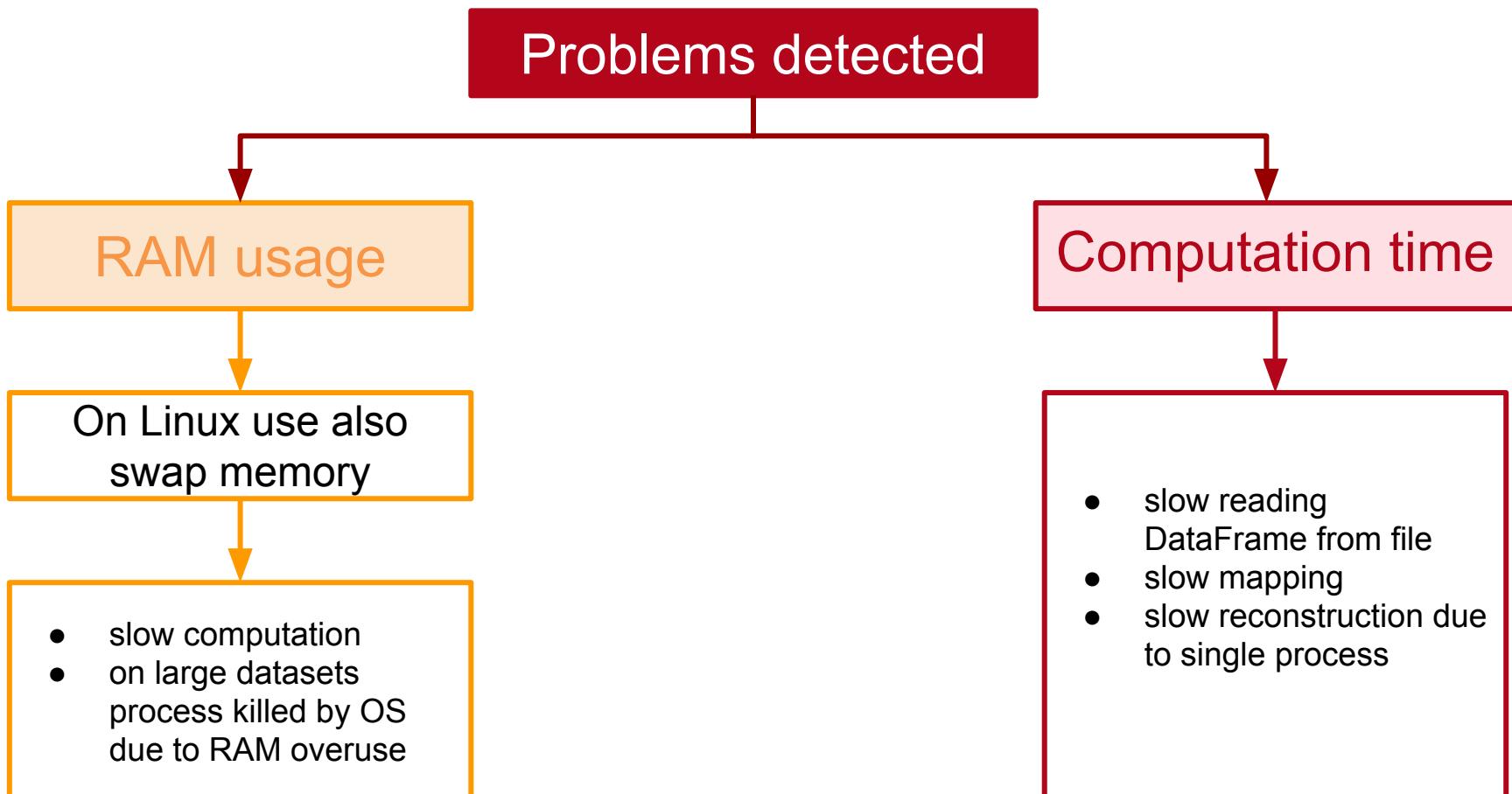
    return
```



Optimization



Problems



Let's analyze each problematic module





mappings.py *not optimized*

```
# set the wire num inside the layer: ranging from 1 to 16 (left to right)
# in tdc_channel_norm the channel is normalized from 0->63 for each sl
df["TDC_CHANNEL_NORM"] = df["TDC_CHANNEL"] % 64 # .astype(np.uint8)
df["WIRE_NUM"] = df["TDC_CHANNEL_NORM"] // 4 + 1 # .astype(np.uint8)

# assign the wire position
for layer in [1, 2, 3, 4]:
    # local wire x position
    df.loc[df["LAYER"] == layer, "WIRE_X_LOC"] = (
        df["WIRE_NUM"] - 1
    ) * XCELL + X_POS_SHIFT[layer]

    # local wire z position
    df.loc[df["LAYER"] == layer, "WIRE_Z_LOC"] = Z_POS_SHIFT[layer]

df = df.astype({"SL": "int8", "LAYER": "int8"})
return df
```

useless columns in DataFrame



more RAM occupied

conversion to uint8 as last operation



deal with bigger size DataFrame
in for loop



mappings.py optimized

```
df = df.astype({"SL": "uint8", "LAYER": "uint8"})
# set the wire num inside the layer: ranging from 1 to 16 (left to right)
# tdc_channel is normalized from 0->63 for each sl
# assign the wire position
for layer in [1, 2, 3, 4]:
    # local wire x position
    df.loc[df["LAYER"] == layer, "WIRE_X_LOC"] = (
        df["TDC_CHANNEL"] % 64 // 4
    ) * XCELL + X_POS_SHIFT[layer]

    # local wire z position
    df.loc[df["LAYER"] == layer, "WIRE_Z_LOC"] = Z_POS_SHIFT[layer]

return df
```

Not Optimized*:

- RAM: 12.85 MB
- Time: 0.061 s

Optimized*:

- RAM: 8.71 MB
- Time: 0.066 s

*On 1/3 of 000054 dataset



EventsFactory.py not optimized

```
def getEvents(stream_df, cfg, runTimeShift):  
  
    # create a dataframe with only valid hits->trig words and scint hits removed  
    hits_df = stream_df[  
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &  
        (stream_df.TDC_CHANNEL <= 127)].copy()  
  
    # fix TDC_MEAS data type  
    hits_df = hits_df.dropna()  
    hits_df = hits_df[  
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &  
        (stream_df.TDC_CHANNEL <= 127)].copy()  
  
    # map hit position  
    hits_df = map_hit_position(hits_df, local=False)  
  
    trigger_df = stream_df[  
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &  
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &  
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"])].copy()  
  
    # create a T0 column (in ns)  
    trigger_df["T0"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30)  
  
    # select only hits in the same orbit of a scint trigger signal  
    hits_df = hits_df.merge(  
        trigger_df, left_on="T0", right_on="T0")  
  
    trigger_df = stream_df[  
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &  
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &  
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"])].copy()  
  
    # calculate T0_NS  
    hits_df_["T0_NS"] = hits_df_[ "T0"] * DURATION_BX  
  
    for sl, offset_sl in cfg["time_offset_sl"].items():  
        # correction is in the form:  
        # coarse offset valid for all SL + fine tuning  
        hits_df_.loc[  
            (hits_df_["SL"] == sl, "T0_NS")  
        ] -= cfg["time_offset_scint"] - runTimeShift + offset_sl # 6 for 123, 8 for 0  
  
    hits_df_ = map_hit_position(hits_df_, local=False)  
  
    return computeEvents(hits_df_)
```

create a copy of a subset of a DataFrame

more RAM occupied.
Can be avoided?



EventsFactory.py not optimized

```
def getEvents(stream_df, cfg, runTimeShift):

    # create a dataframe with only valid hits->trig words and scint hits removed
    hits_df = stream_df[
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &
        (stream_df.TDC_CHANNEL <= 127)].copy()

    # fix TDC_MEAS data type
    hits_df = hits_df.dropna()
    hits_df = hits_df.astype({"TDC_MEAS": "int32"})

    # create mapping with the loaded configurations
    mapper = Mapping(cfg)
    hits_df = mapper.global_map(hits_df)

    # select all orbits with a trigger signal from
    # the scintillators coincidence
    hits_df = hits_df.dropna()
    hits_df = hits_df.astype({"TDC_MEAS": "int32"})

    trigger_dfl["t0"] = (trigger_dfl.BX_COUNTER * trigger_dfl.TDC_MEAS) / 50

    # select only hits in the same orbit of a scint trigger signal
    hits_df_ = pd.merge(
        hits_df, trigger_dfl[["ORBIT_CNT", "T0"]],
        left_on="ORBIT_CNT", right_on="ORBIT_CNT",
        suffixes=(None, None))
    # print the number of valid hits found in data
    print(f"Valid hits: {hits_df_.shape[0]}")

    # create a time column in NS
    hits_df_[ "T0_NS" ] = hits_df_[ "T0" ] * DURATION_BX

    for sl, offset_sl in cfg["time_offset_sl"].items():
        # correction is in the form:
        # coarse offset valid for all SL + fine tuning
        hits_df_.loc[
            hits_df_[ "SL" ] == sl, "T0_NS"]
        ] -= cfg["time_offset_scint"] - runTimeShift + offset_sl # 6 for 123, 8 for 0

    hits_df_ = map_hit_position(hits_df_, local=False)

return computeEvents(hits_df_)
```

.dropna() method creates a new DataFrame;
.astype() requires time to resize memory blocks for dataset

more computation time and RAM usage.
Can be done at reading from file time?



EventsFactory.py *not optimized*

```
def getEvents(stream_df, cfg, runTimeShift):

    # create a dataframe with only valid hits->trig words and scint hits removed
    hits_df = stream_df[
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &
        (stream_df.TDC_CHANNEL <= 127)].copy()

    # fix TDC_MEAS data type
    hits_df = hits_df.dropna()
    hits_df = hits_df.astype({"TDC_MEAS": "int32"})

    # create mapping with the loaded configurations
    mapper = Mapping(cfg)
    hits_df = mapper.global_map(hits_df)

    # select all objects with a trigger signal from
    # the scintillators coincidence
    trigger_df = stream_df[
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &

# create mapping with the loaded configurations
mapper = Mapping(cfg)
hits_df = mapper.global_map(hits_df)

# print the number of valid hits found in data
print(f"Valid hits: {hits_df.shape[0]}")

# create a time column in NS
hits_df["T0_NS"] = hits_df["T0"] * DURATION_BX

for sl, offset_sl in cfg["time_offset_sl"].items():
    # correction is in the form:
    # coarse offset valid for all SL + fine tuning
    hits_df.loc[
        hits_df["SL"] == sl, "T0_NS"
    ] -= cfg["time_offset_scint"] - runTimeShift + offset_sl # 6 for 123, 8 for 0

    hits_df_ = map_hit_position(hits_df_, local=False)

return computeEvents(hits_df_)
```

mapping on DataFrame not completely filtered yet

compute variables on useless rows.
Can be avoided?



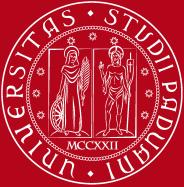
EventsFactory.py not optimized

```
def getEvents(stream_df, cfg, runTimeShift):  
  
    # create a dataframe with only valid hits->trig words and scint hits removed  
    hits_df = stream_df[  
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &  
        (stream_df.TDC_CHANNEL <= 127)].copy()  
  
    # create a T0 column (in ns)  
    trigger_df["T0"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30)  
  
    # select all orbits with a trigger signal from  
    # the scintillators coincidence  
    trigger_df = stream_df[  
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &  
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &  
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"]).copy()  
  
    # create a T0 column (in ns)  
    trigger_df["T0"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30)  
  
    # select only hits in the same orbit of a scint trigger signal  
    hits_df_ = pd.merge(  
        hits_df, trigger_df[["ORBIT_CNT", "T0"]],  
        left_on="ORBIT_CNT", right_on="ORBIT_CNT",  
        suffixes=(None, None)  
    )  
    # print the number of valid hits found in data  
    print(f"Valid hits: {hits_df_.shape[0]}")  
  
    # create a time column in NS  
    hits_df_[ "T0_NS" ] = hits_df_[ "T0" ] * DURATION_BX  
  
    for sl, offset_sl in cfg["time_offset_sl"].items():  
        # correction is in the form:  
  
        # create a time column in NS  
        hits_df_[ "T0_NS" ] = hits_df_[ "T0" ] * DURATION_BX
```

useless new column on DataFrame



more RAM occupied



EventsFactory.py not optimized

```
def getEvents(stream_df, cfg, runTimeShift):  
  
    # create a dataframe with only valid hits->trig words and scint hits removed  
  
    def computeEvents(hits_df_):  
        events = []  
        for x in pd.unique(hits_df_.ORBIT_CNT):  
            events.append(hits_df_[hits_df_["ORBIT_CNT"] == x])  
            events[-1] = events[-1].reset_index(drop=True)  
        return events  
  
    trigger_df = stream_df[  
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &  
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &  
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"])].copy()  
  
    # create a T0 column (in ns)  
    trigger_df["T0"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30)  
  
    # select only hits in the same orbit of a scint trigger signal  
    hits_df_ = pd.merge(  
        hits_df, trigger_df[["ORBIT_CNT", "T0"]],  
        left_on="ORBIT_CNT", right_on="ORBIT_CNT",  
        suffixes=(None, None)  
    )  
    # print the number of valid hits found in data  
    print(f"Valid hits: {hits_df_.shape[0]}")  
  
    # create a time column in NS  
    hits_df_[["T0_NS"]] = hits_df_[["T0"]] * DURATION_BX  
  
    for sl, offset_sl in cfg["time_offset_sl"].items():  
        # correction is in the form:  
        # coarse offset valid for all SL or fine tuning  
        hits_df_.loc[  
            hits_df_[["SL"]] == sl, "T0_NS"  
        ] -= cfg["time_offset_scint"] - runTimeShift + offset_sl # 6 for 123, 8 for 0  
  
    hits_df_ = map_hit_position(hits_df_, local=False)  
  
    return computeEvents(hits_df_)
```

use of non-pandas methods

more RAM occupied and time consuming operations



EventsFactory.py optimized

```
def getEvents(df_fname, cfg, runTimeShift):
    #reading df from file
    dtype_dict = { 'HEAD':np.uint8, 'FPGA':np.uint8, 'TDC_CHANNEL':np.uint8,
                   'ORBIT_CNT':np.uint64, 'BX_COUNTER':np.uint16, 'TDC_MEAS':np.uint8 }
    print("Reading dataset from file...")
    stream_df = pd.read_csv(df_fname, dtype=dtype_dict)
    #drop NaN
    stream_df.dropna(inplace=True)

    # create a dataframe with only valid hits-trig words and scint hits removed
    hits_df = stream_df[
        (stream_df['HEAD'] == cfg["header"]["valid_hit"]) &
        (stream_df['TDC_CHANNEL'] <= 127)]  
  
    # select all orbits with a trigger signal from
    # the scintillators coincidences
    trigger_df = stream_df[
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"])]  
  
    # create a T0 column (in ns)
    trigger_df["T0_NS"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30) * DS  
RATION_BX
    # select only hits in the same orbit of a scint trigger signal
    hits_df_ = pd.merge(
        hits_df, trigger_df[['ORBIT_CNT', "T0_NS"]],
        left_on="ORBIT_CNT", right_on="ORBIT_CNT",
        suffixes=(None, None))
    )  
  
    del trigger_df
    del hits_df_
    #print the number of valid hits found in data
    print(f"Valid hits: {hits_df_.shape[0]}")  
  
    del stream_df
    # create mapping with the loaded configurations
    mappings = Mapping(cfg)

    #reading df from file
    dtype_dict = { 'HEAD':np.uint8, 'FPGA':np.uint8, 'TDC_CHANNEL':np.uint8,
                   'ORBIT_CNT':np.uint64, 'BX_COUNTER':np.uint16, 'TDC_MEAS':np.uint8 }
    print("Reading dataset from file...")
    stream_df = pd.read_csv(df_fname, dtype=dtype_dict)
    #drop NaN
    stream_df.dropna(inplace=True)

    return computeEvents(hits_df_)
```

Not Optimized*:

- RAM: 327 MB
- Time: 1 s

Optimized*:

- RAM: 123 MB
- Time: 0.09 s

*On 1/3 of 000054 dataset



EventsFactory.py optimized

```
def getEvents(df_fname, cfg, runTimeShift):
    #reading df from file
    dtype_dict = { 'HEAD':np.uint8, 'FPGA':np.uint8, 'TDC_CHANNEL':np.uint8,
    'ORBIT_CNT':np.uint64, 'BX_COUNTER':np.uint16, 'TDC_MEAS':np.uint8 }
    print("Reading dataset from file...")
    stream_df = pd.read_csv(df_fname, dtype=dtype_dict)
    #drop NaN
    stream_df.dropna(inplace=True)

    # create a dataframe with only valid hits->trig words and scint hits removed
    hits_df = stream_df[
        (stream_df.HEAD == cfg["headers"]["valid_hit"]) &
        (stream_df.TDC_CHANNEL <= 127)]

    # select all orbits with a trigger signal from
    # the scintillators coincidence
    trigger_df = stream_df[
        (stream_df["HEAD"] == cfg["scintillator"]["head"]) &
        (stream_df["FPGA"] == cfg["scintillator"]["fpga"]) &
        (stream_df["TDC_CHANNEL"] == cfg["scintillator"]["tdc_ch"])]
```

calculate directly T0_NS and use it to do the merge()

```
    # create a T0 column (in ns)
    trigger_df["T0_NS"] = (trigger_df["BX_COUNTER"] + trigger_df["TDC_MEAS"] / 30) * DU
    RATION_BX
    # select only hits in the same orbit of a scint trigger signal
    hits_df_ = pd.merge(
        hits_df, trigger_df[["ORBIT_CNT", "T0_NS"]],
        left_on="ORBIT_CNT", right_on="ORBIT_CNT",
        suffixes=(None, None))
    )

    del trigger_df
    del hits_df
    # print the number of valid hits found in data
    print(f"Valid hits: {hits_df_.shape[0]}")

    del stream_df
    # create mapping with the loaded configurations
    mapper = Mapping(cfg)
    # map hits
    hits_df_ = mapper.global_map(hits_df_)

#TIME SHIFTING
for sl, offset_sl in c:
    # correction is in ns
    # coarse offset via
    hits_df_.loc[
        hits_df_["SL"] == sl]
    ] -= cfg["time_offset"]
    hits_df_ = map_hits_offset(hits_df_, offset_sl)

return computeEvents(hits_df_)
```

Moved Mapping after filtering, compute variables over smaller DataFrame

```
def computeEvents(hits_df_):
    events = [group for _, group in hits_df_.groupby("ORBIT_CNT")
              if len(group) <= 32]
    return events
```

Not Optimized*:

- RAM: 486 MB
- Time: 7.9 s

Optimized*:

- RAM: 252 MB
- Time: 0.7 s

*On 1/3 of 000054 dataset



reco.py not optimized compute()

```
def compute(df):

    comb = []
    if len(df.LAYER.unique()) == 3:
        comb.append(df)
        tot_Hits = 3
    else:
        for index in list(combinations(df.index, 4)):
            tmp_df = df.loc[index, :]
            if len(tmp_df.LAYER.unique()) == 4:
                comb.append(tmp_df) # comb[] contains combinations of data
        tot_Hits = 4
    # saving ORBIT_CNT
    orbit = np.array(df["ORBIT_CNT"])[0]
    # saving SL
    sl = np.array(df["SL"])[0]

    flag = True
    for data in comb:
        X = np.array(pd.concat([data["X_RIGHT_GLOB"], data["X_LEFT_GLOB"]]))
        Y = np.array(pd.concat([data["WIRE_Z_GLOB"], data["WIRE_Z_GLOB"]]))
        for indexes_comb in list(combinations(range(len(X)), tot_Hits)):
            new_X = []
            new_Y = []
            for i in indexes_comb:
                new_X.append(X[i])
                new_Y.append(Y[i])

            if len(np.unique(new_Y)) == tot_Hits:
                regr_tuple = linear_regr(new_Y)
                if flag:
                    min_lambda = abs(regr_tuple["chisq_comp"])
                    xdata = new_X
                    ydata = new_Y
                    res_dict = regr_tuple
                    flag = False
                    best_comb = indexes_comb
                    best_data = data
                    best_data = data
                elif abs(regr_tuple["chisq_comp"]) < min_lambda:
                    best_comb = indexes_comb
                    min_lambda = abs(regr_tuple["chisq_comp"])
                    xdata = new_X
                    ydata = new_Y
                    res_dict = regr_tuple
                    best_data = data

    big_df = pd.concat([best_data, best_data], axis=0, ignore_index=True)
    reco_df = big_df.loc[best_comb, :]
    reco_df["m"] = np.full(len(reco_df), res_dict["m"])
    reco_df["q"] = np.full(len(reco_df), res_dict["q"])
    reco_df["X"] = xdata
    res_dict["ORBIT_CNT"] = orbit
    res_dict["SL"] = sl
    return res_dict, xdata, ydata, reco_df
```

new array created at each iteration

too long code, can be reduced

useless DataFrame created



reco.py optimized compute()

```
def compute(df):

    comb = []
    if len(df.LAYER.unique()) == 3:
        comb.append(df)
        tot_Hits = 3
    else:
        for index in list(combinations(df.index, 4)):
            if len(df.loc[index, :].LAYER.unique()) == 4:
                comb.append(df.loc[index, :])
        tot_Hits = 4

    min_lambda = np.finfo(float).max

    for data in comb:
        X = np.array(pd.concat([data["X_RIGHT_GLOB"], data["X_LEFT_GLOB"]]))
        Y = np.array(pd.concat([data["WIRE_Z_GLOB"], data["WIRE_Z_GLOB"]]))
        for indexes_comb in list(combinations(range(len(X)), tot_Hits)):
            indexes_comb = list(indexes_comb)
            if len(np.unique(Y[indexes_comb])) == tot_Hits:
                regr_dict = linear_regr(X[indexes_comb], Y[indexes_comb])
                if abs(regr_dict["chisq_comp"]) < min_lambda:
                    min_lambda = abs(regr_dict["chisq_comp"])
                    xdata = X[indexes_comb]
                    res_dict = regr_dict
                    best_comb = indexes_comb
                    best_data = data

    reco_df = pd.concat([best_data, best_data], axis=0, ignore_index=True)
    reco_df = reco_df.loc[best_comb, :]
    reco_df["m"] = np.full(len(reco_df), res_dict["m"])
    reco_df["q"] = np.full(len(reco_df), res_dict["q"])
    reco_df["X"] = xdata
    if xdata is None: return

    return reco_df
```

used slicing instead
of creating new
array

reduced code

no more useless
DataFrame

Not Optimized*:

- RAM: 0.09 MB
- Time: 0.04 s

Optimized*:

- RAM: 0.08 MB
- Time: 0.04 s

*On a single event DataFrame



reco.py

getRecoResults()

Not Optimized:

```
def getRecoResults(events):
    resultsDf = []

    for df_E in events:
        event_reco_df = computeEvent(df_E)
        if event_reco_df is None:
            continue
        if len(event_reco_df)==0:
            continue
        resultsDf.append(event_reco_df)

    return resultsDf
```

each reconstruction on a single event DataFrame launched on the same process, only one CPU core used

Not Optimized*:

- RAM: 35 MB
- Time: 436 s

Optimized:

```
def getRecoResults_mp(events):
    pool = Pool(processes=cpu_count()-2)

    result = pool.map_async(computeEvent, events)
    resultsDf = result.get()
    pool.close()
    pool.join()
    resultsDf = [x for x in resultsDf if x is not None]

    return resultsDf
```

more CPU cores are used in order to reconstruct the event through multiprocessing library

Optimized*:

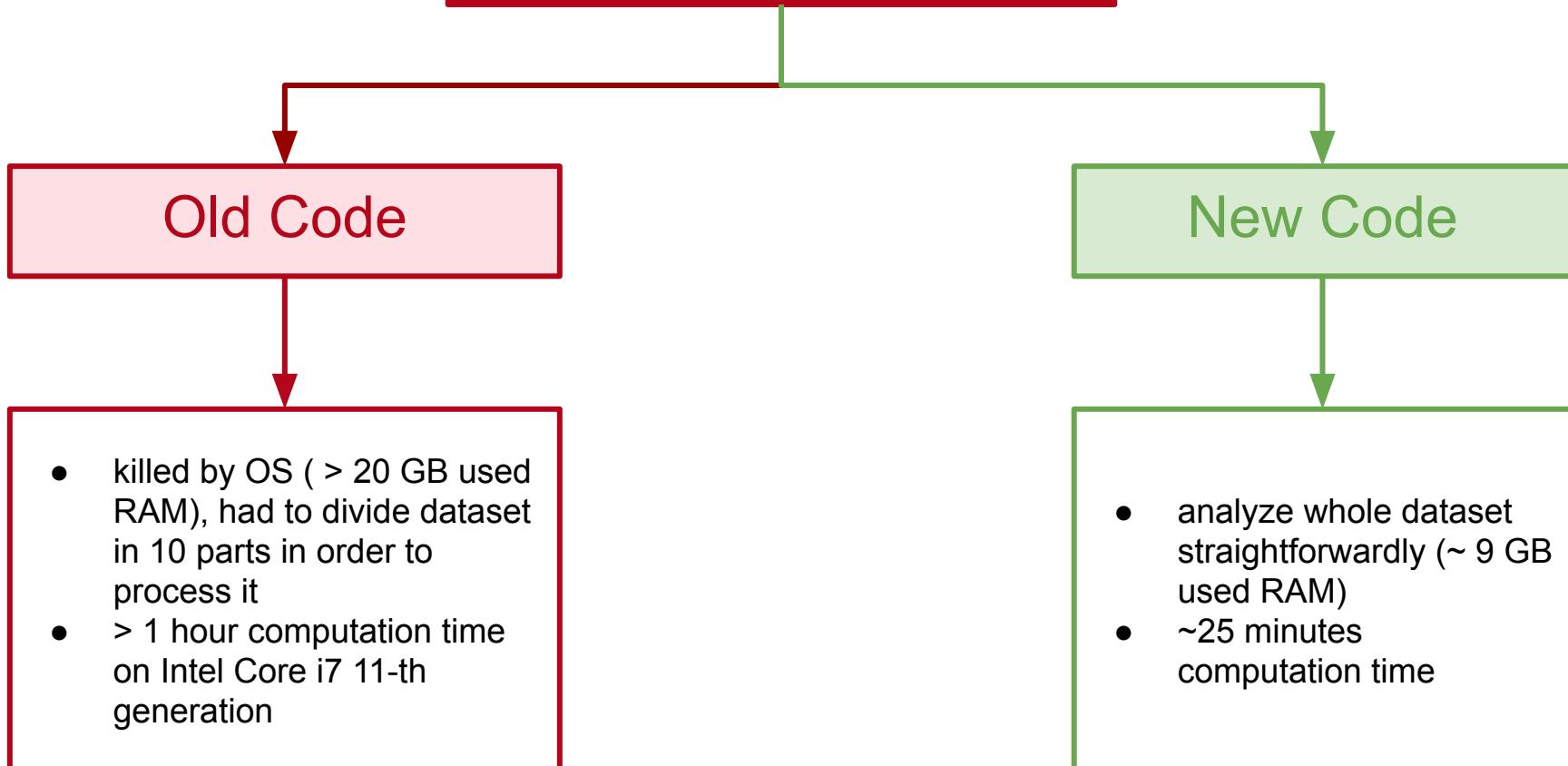
- RAM: 25 MB
- Time: 157 s

*On $\frac{1}{3}$ of 000054 dataset



Results

Results on analysis of 000054 dataset

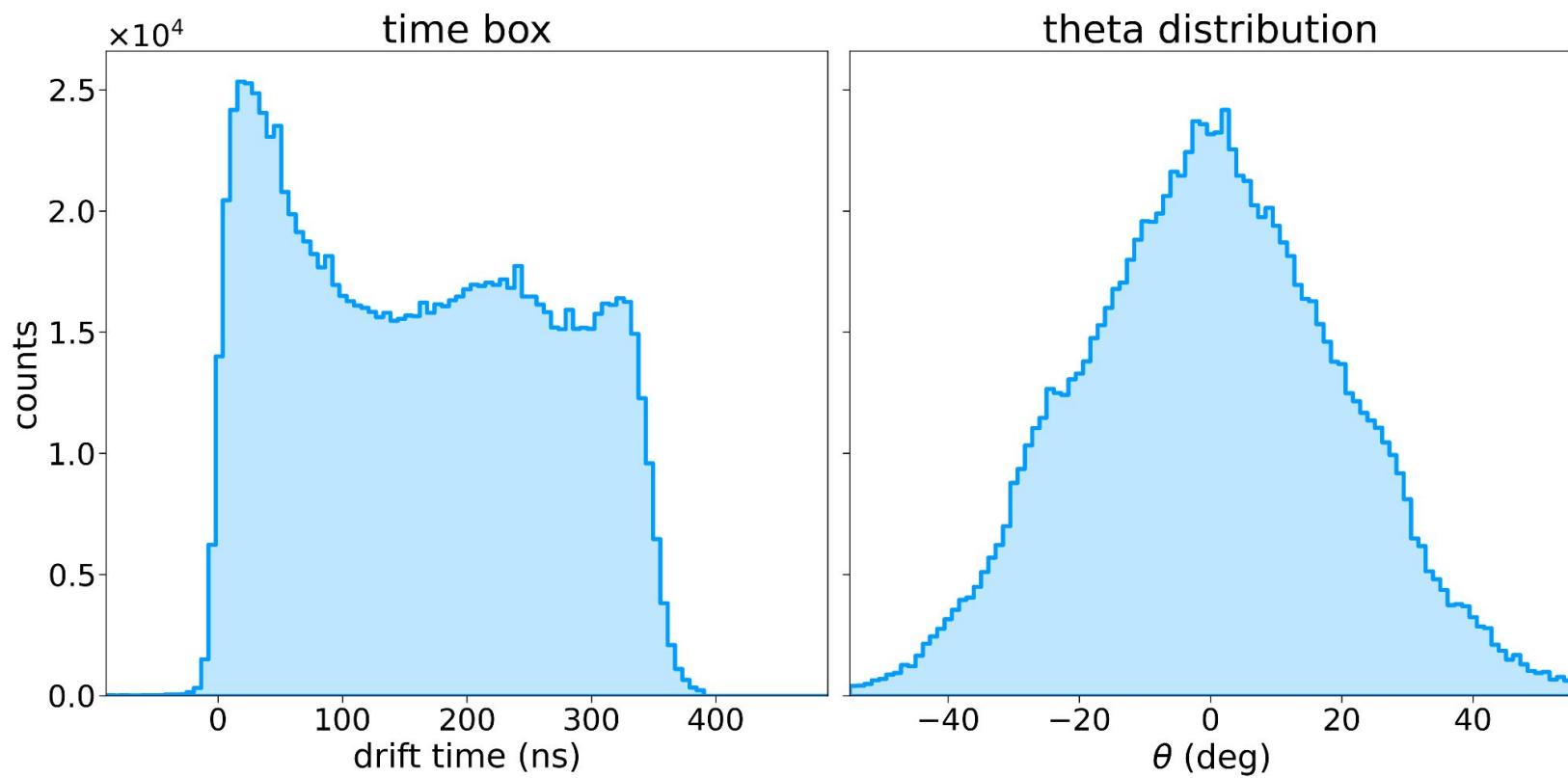


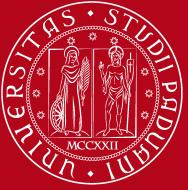


θ - drift time correlation

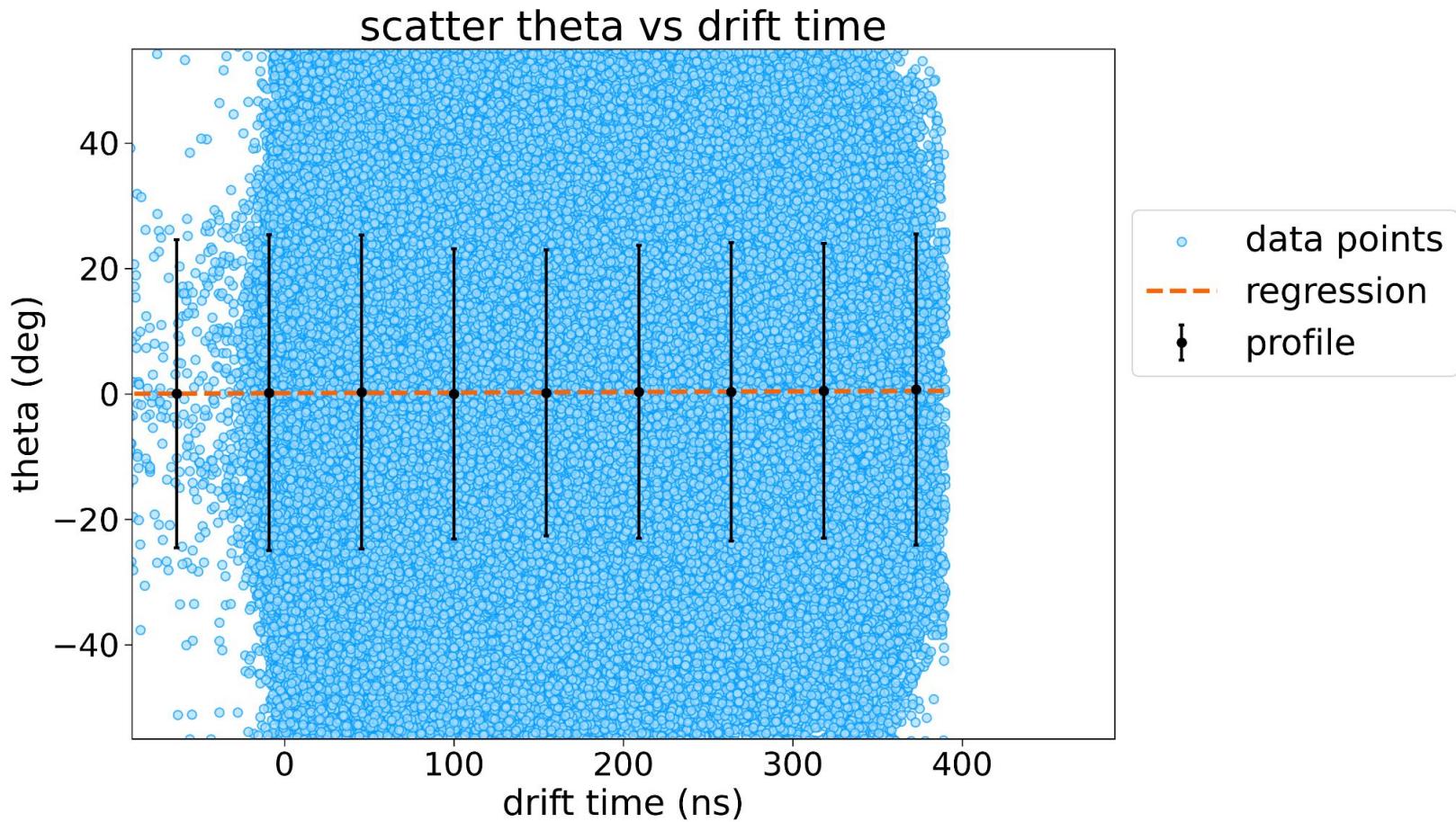


Drift Time and Theta Distribution



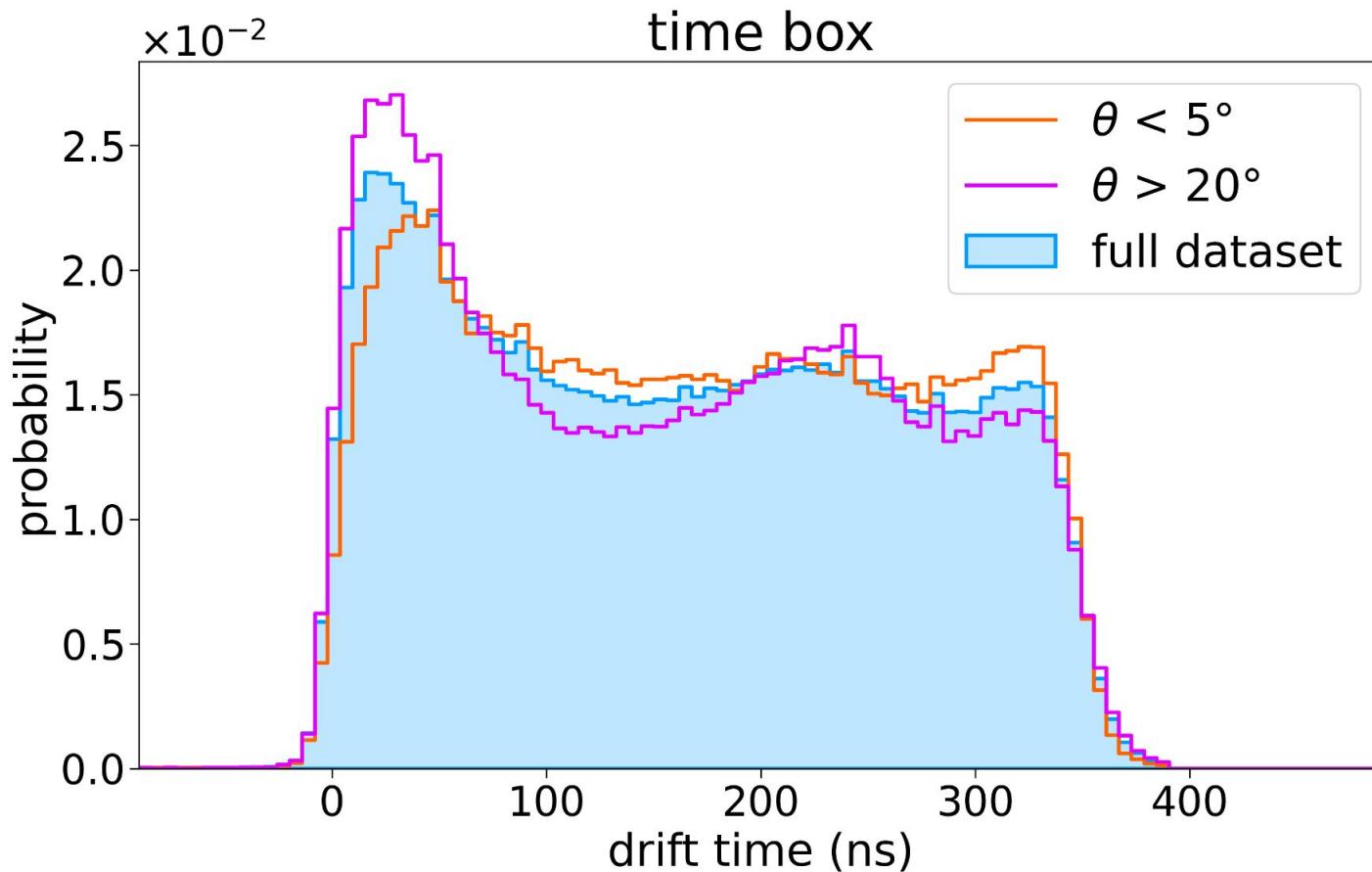


Data Inspection





Start cutting data





Two statistical approaches

Normalization

Probability

sum of all the heights
is equal to 1

for each bin:

$$h_{\text{norm},i} = \frac{h_i}{\sum_{j=0}^{n_{\text{bin}}} h_j}$$

Density

total area = 1

for each bin:

$$h_{\text{norm},i} = \frac{h_i}{\int h(x)dx}$$

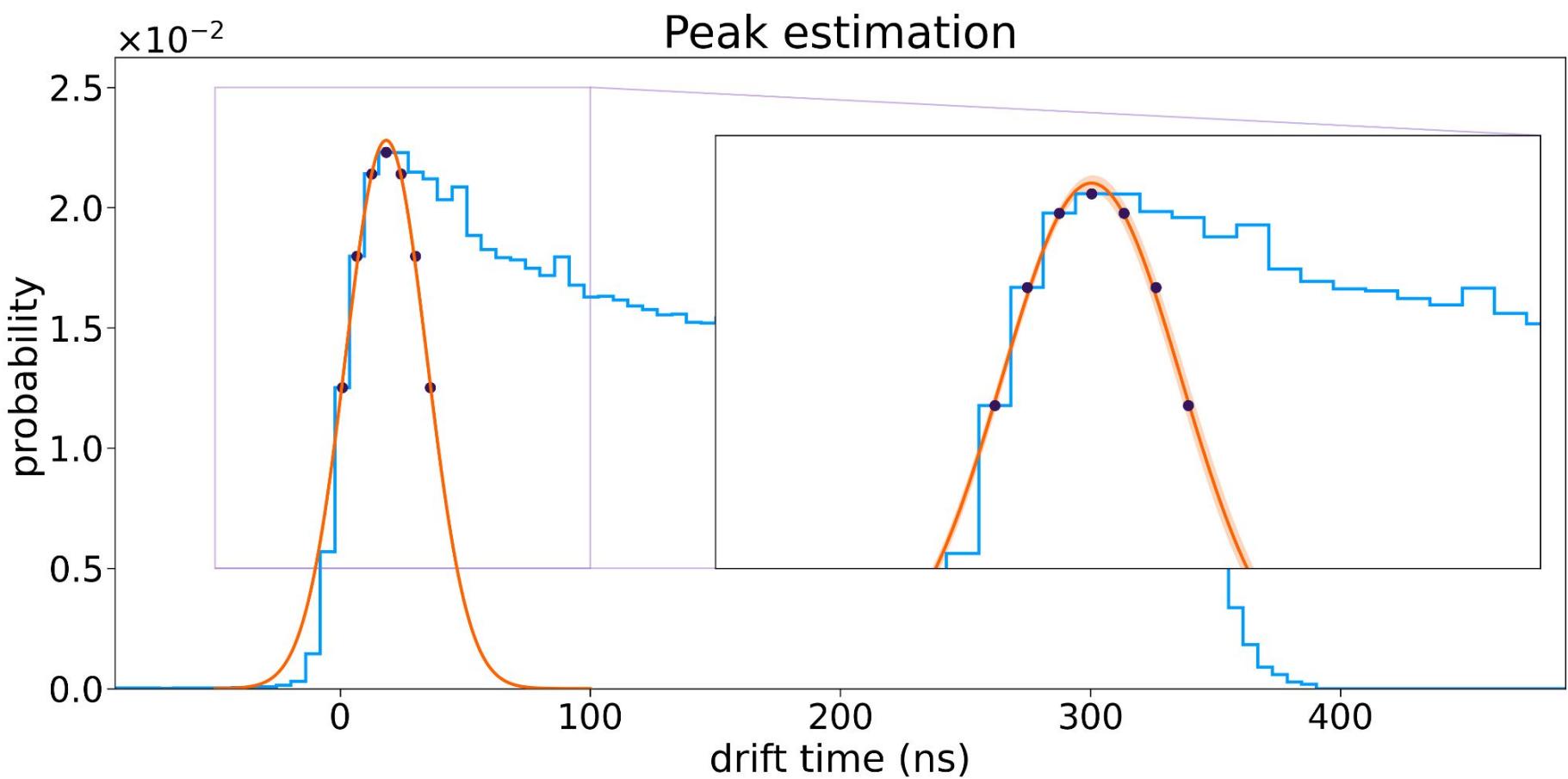


Analysis steps

- start cutting dataset → Theta intervals
 1. low cut : $\theta < \theta_i$
 2. high cut : $\theta > \theta_i$
 3. middle cut : $\theta_{low} < \theta < \theta_{high}$
- estimate the probability of small drift time ($0 \text{ ns} < t < 97 \text{ ns}$) for each cutting approach
- To better estimate the height of the peaks → Gaussian fit

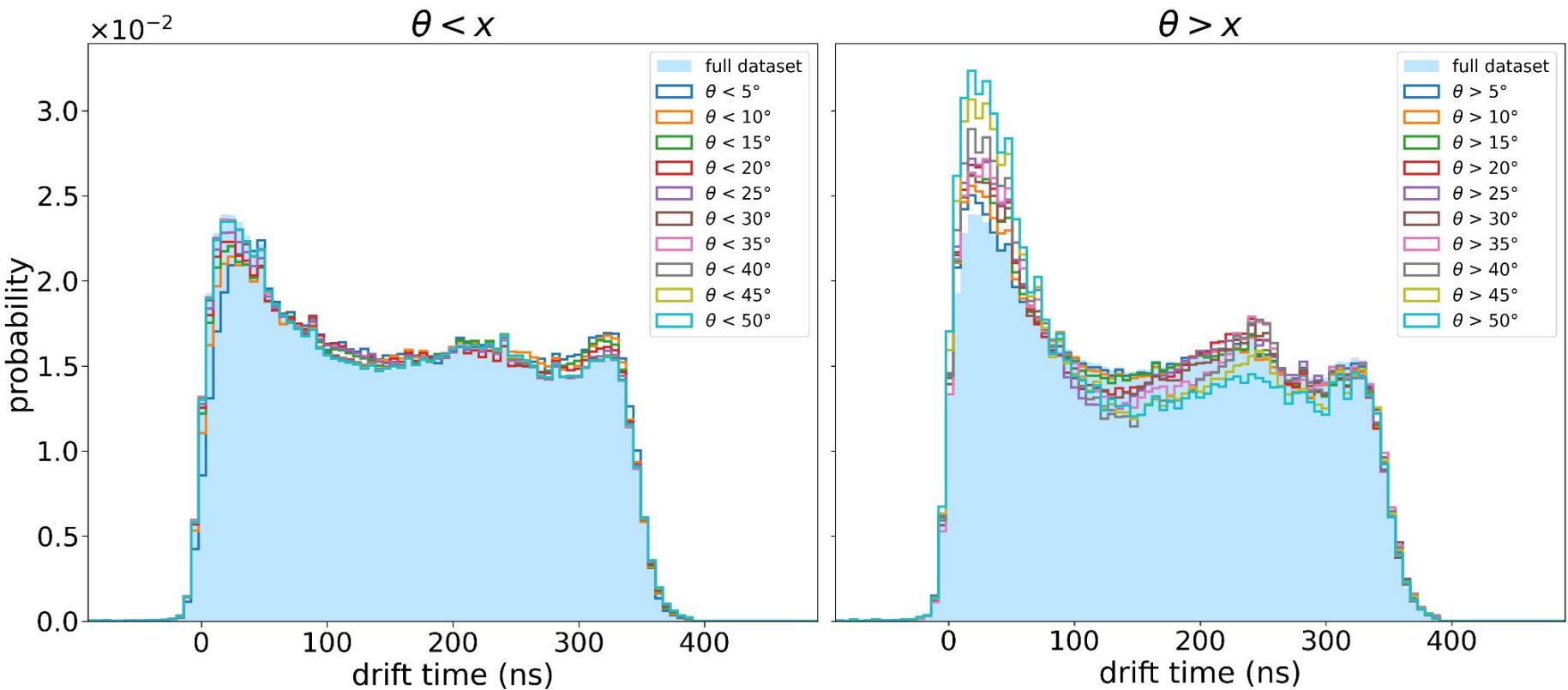


Gaussian fit



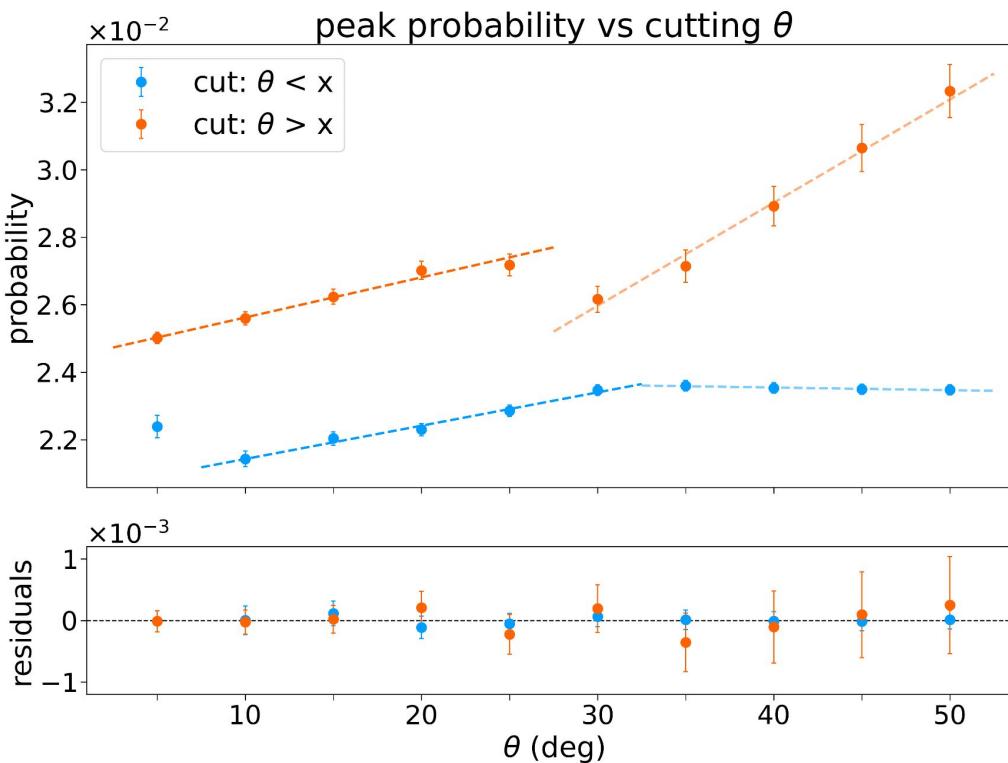


Probability: Low and high cut histograms





Probability: Low and high cut fit



r = 0.99317
p_value = 6.77e-04
chi = 0.9843
p_value_chi = 0.805

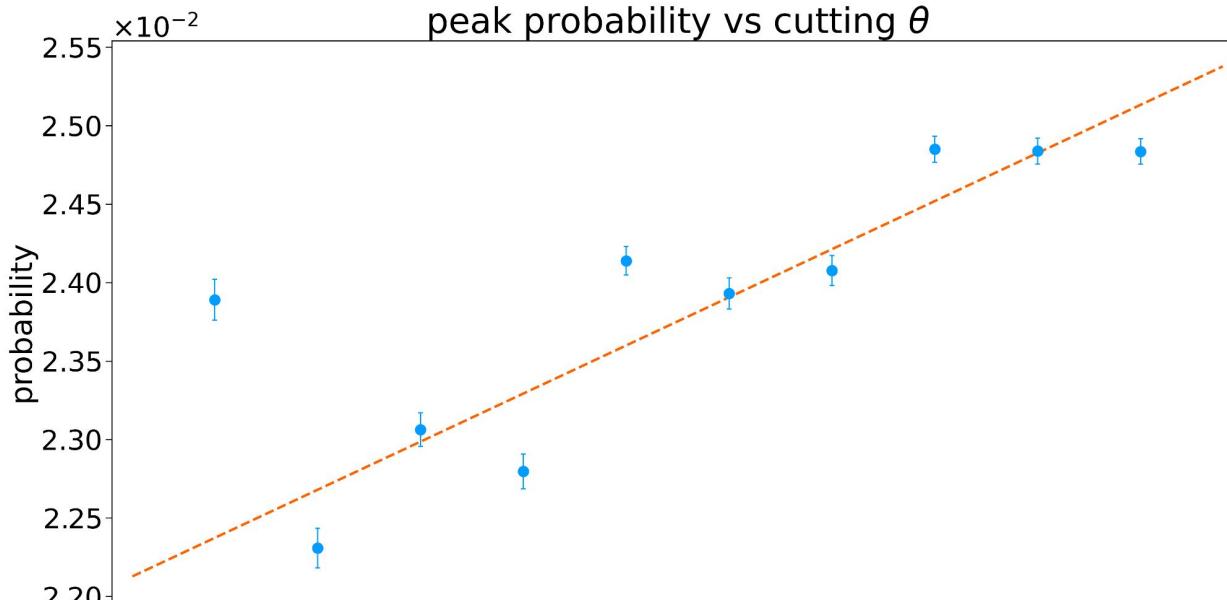
r = 0.9530
p_value = 4.69e-02
chi = 0.0323
p_value_chi = 0.984

r = 0.98644
p_value = 1.89e-03
chi = 1.1326
p_value_chi = 0.769

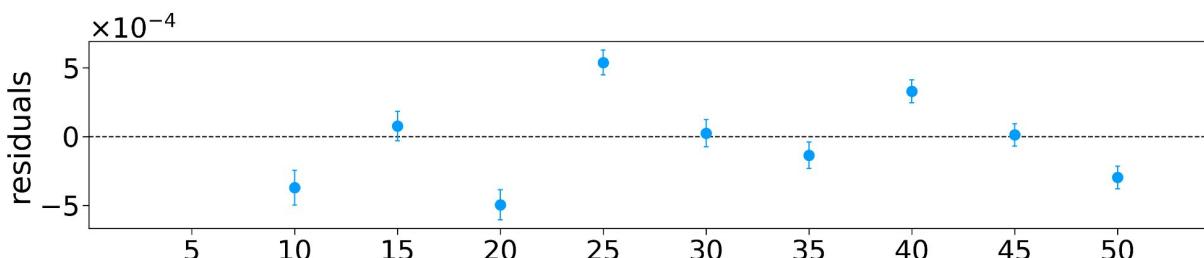
r = 0.99575
p_value = 3.32e-04
chi = 0.96601
p_value_chi = 0.809



Probability: Low cut gaussian

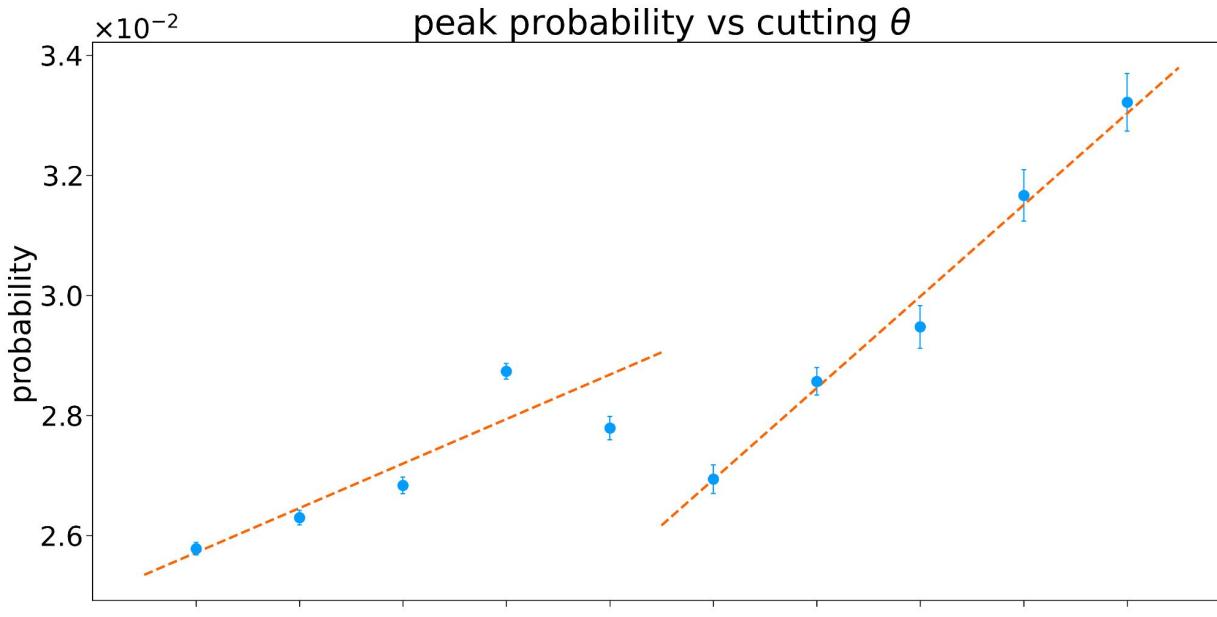


r = 0.93778
p_value = 1.86e-04
chi = 94.8909
p_value_chi = 1.22e-17



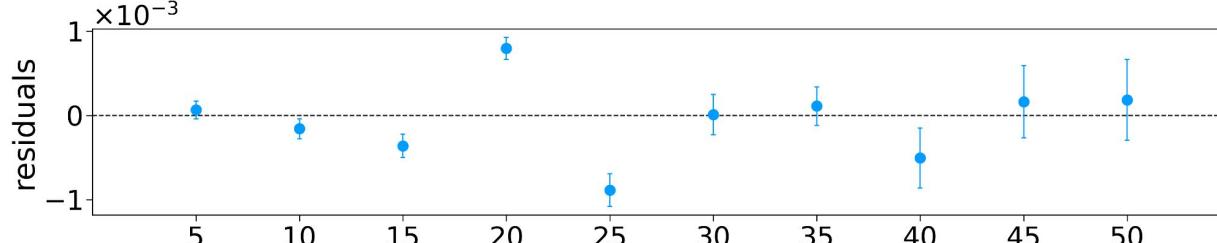


Probability: high cut gaussian



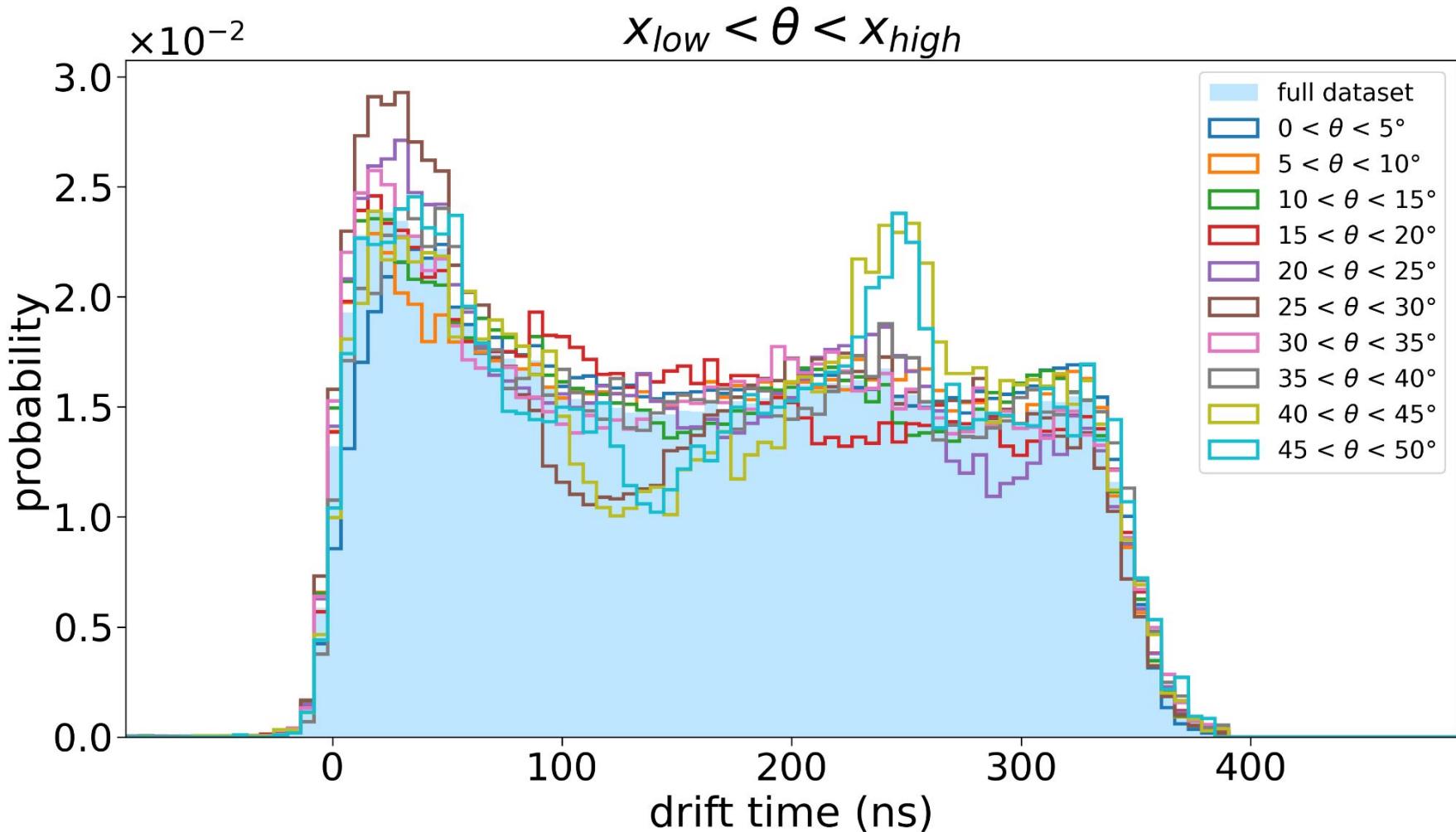
r = 0.86299
p_value = 5.96e-02
chi = 67.515
p_value_chi = 1.45e-14

r = 0.99369
p_value = 6.02e-04
chi = 2.5304
p_value_chi = 0.4698



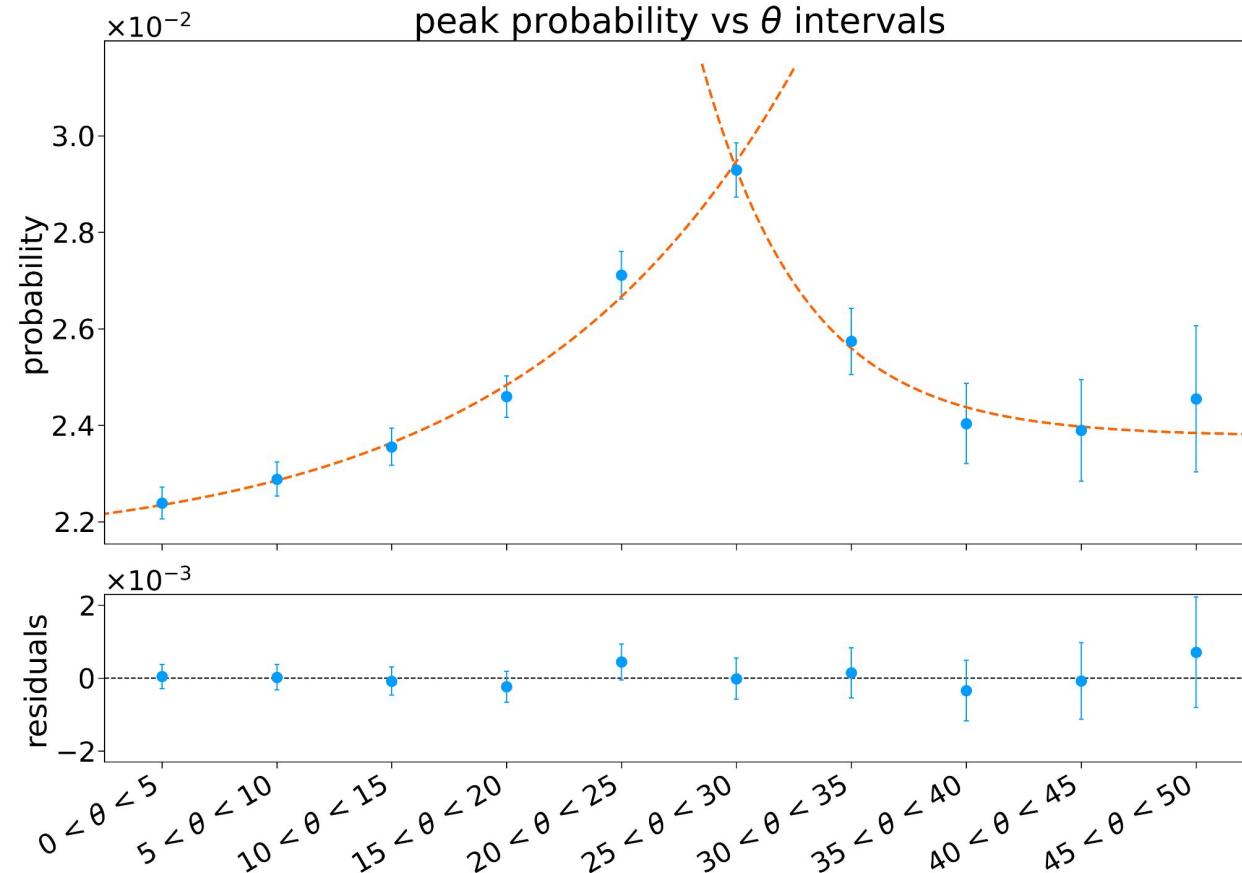


Probability: Middle cut histograms





Probability: Middle cut fit

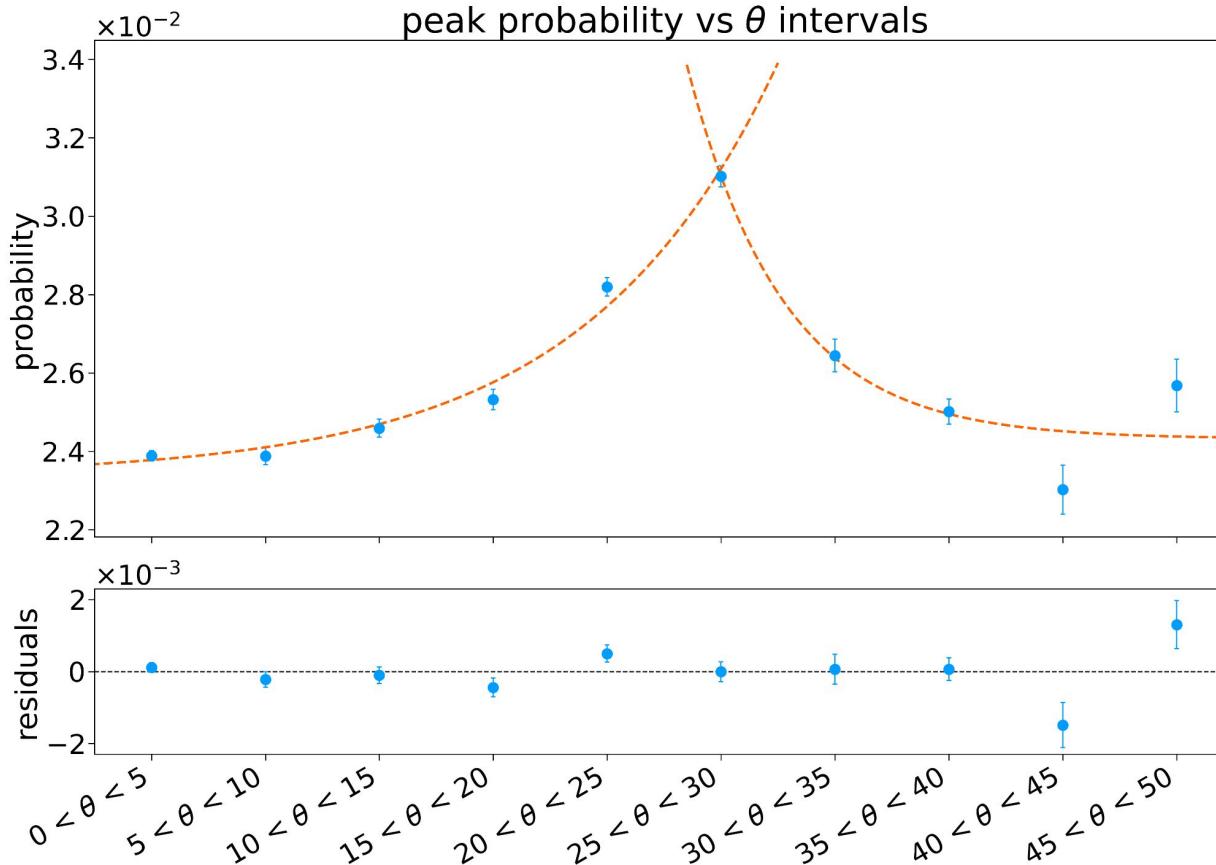


chi = 1.2911
p_value_chi = 0.7312

chi = 0.4374
p_value_chi = 0.804



Probability: Middle cut gaussian

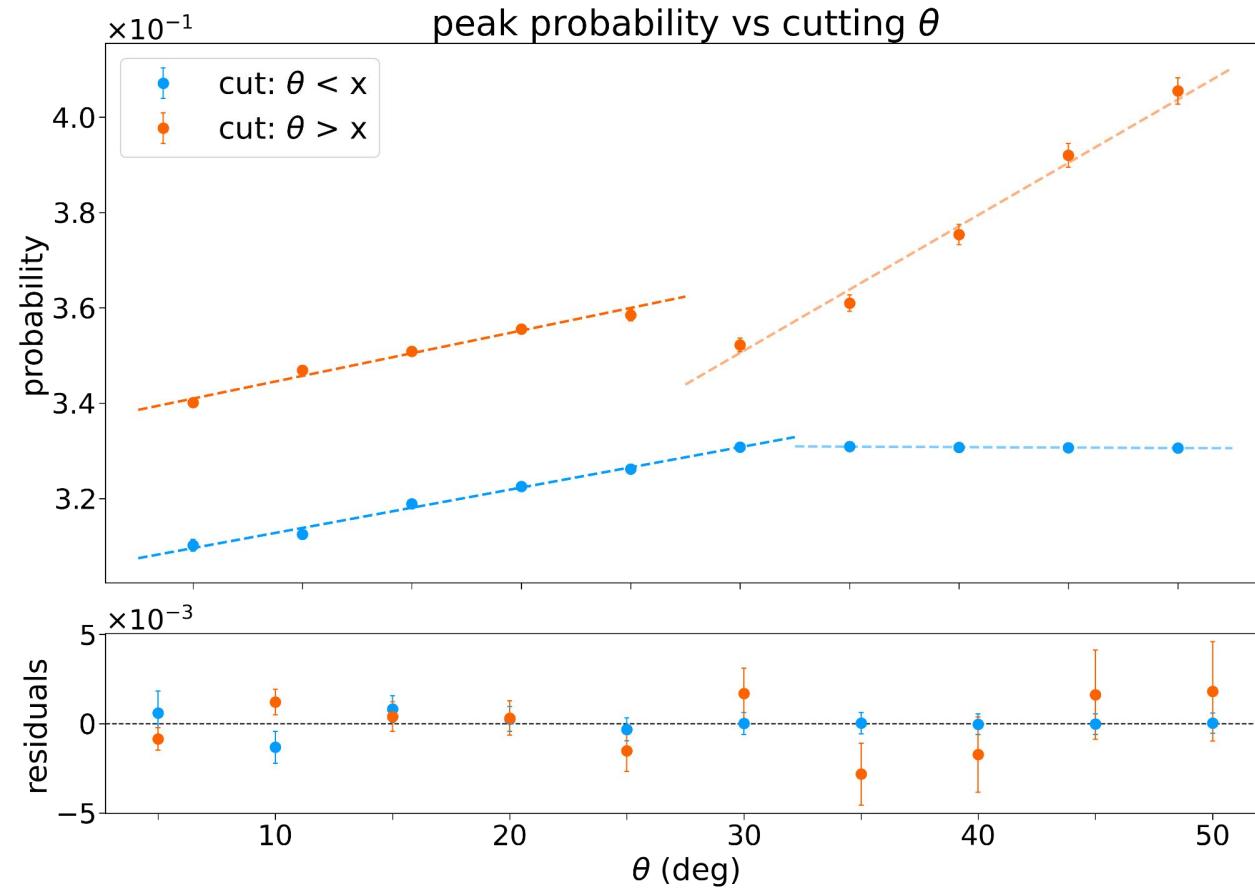


chi = 9.4783
p_value_chi = 0.0207

chi = 9.7592
p_value_chi = 1.453e-14



Density: Low and high cut fit



r = 0.9953
p_value = 3.27e-05
chi = 3.936
p_value_chi = 0.415

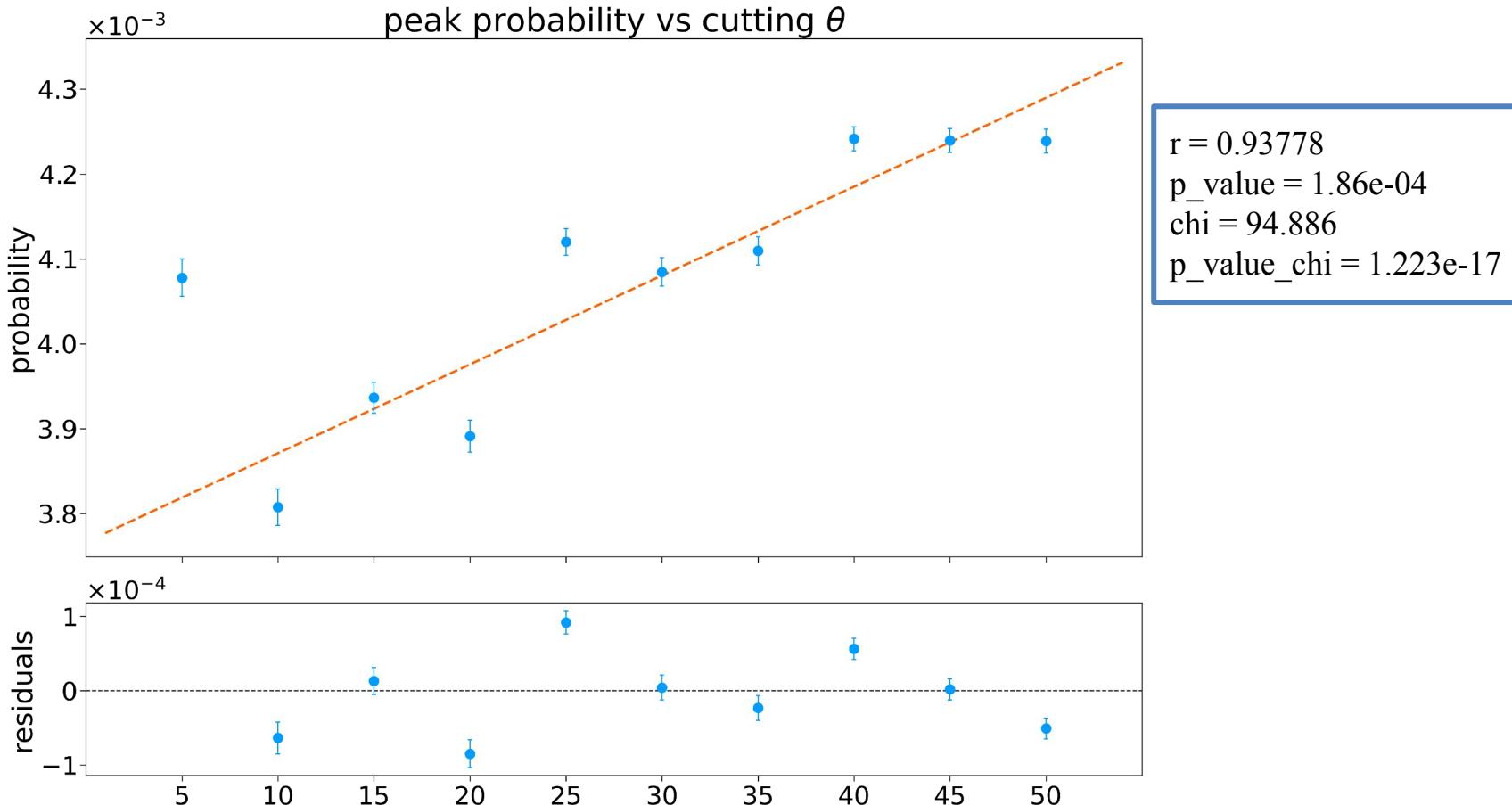
r = 0.96354
p_value = 3.65e-02
chi = 0.0108
p_value_chi = 0.995

r = 0.9898
p_value = 1.22e-03
chi = 6.7043
p_value_chi = 0.152

r = 0.9954
p_value = 3.72e-04
chi = 5.5726
p_value_chi = 0.134

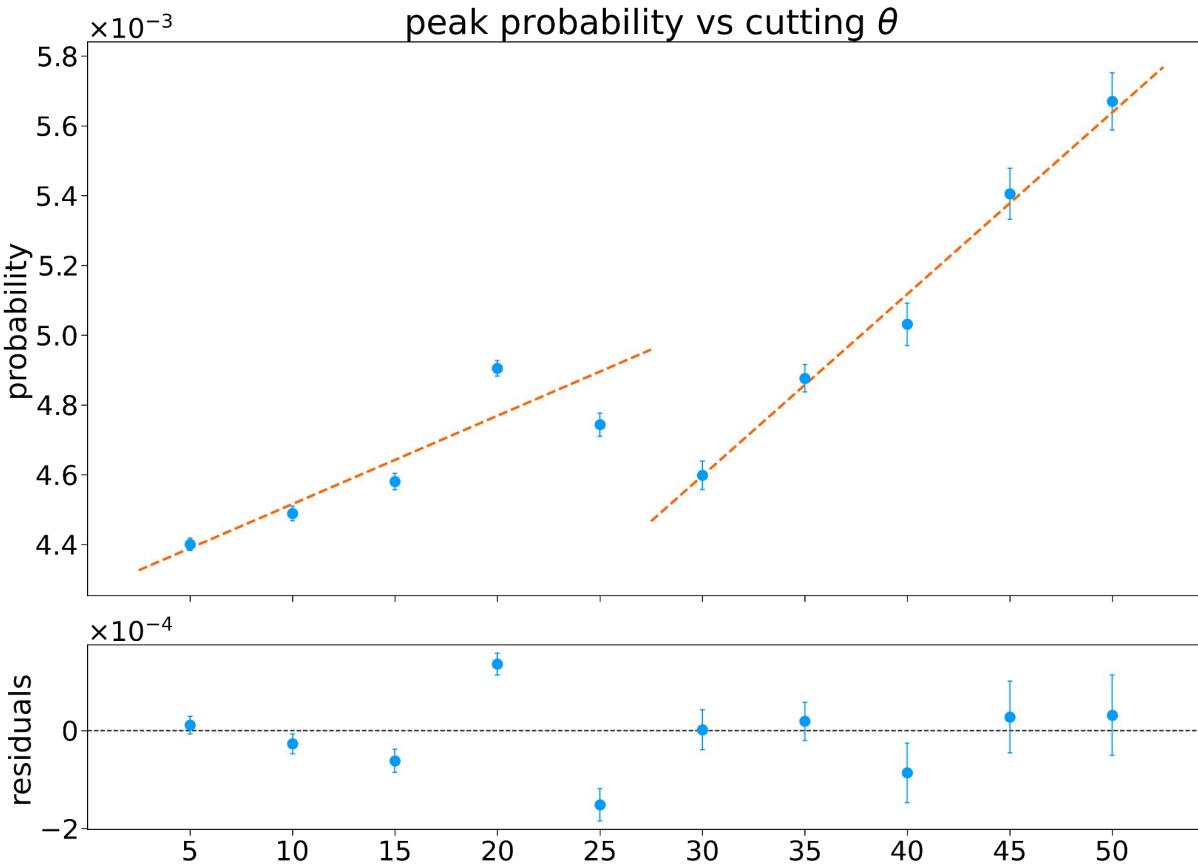


Density: Low cut gaussian





Density: Low high gaussian

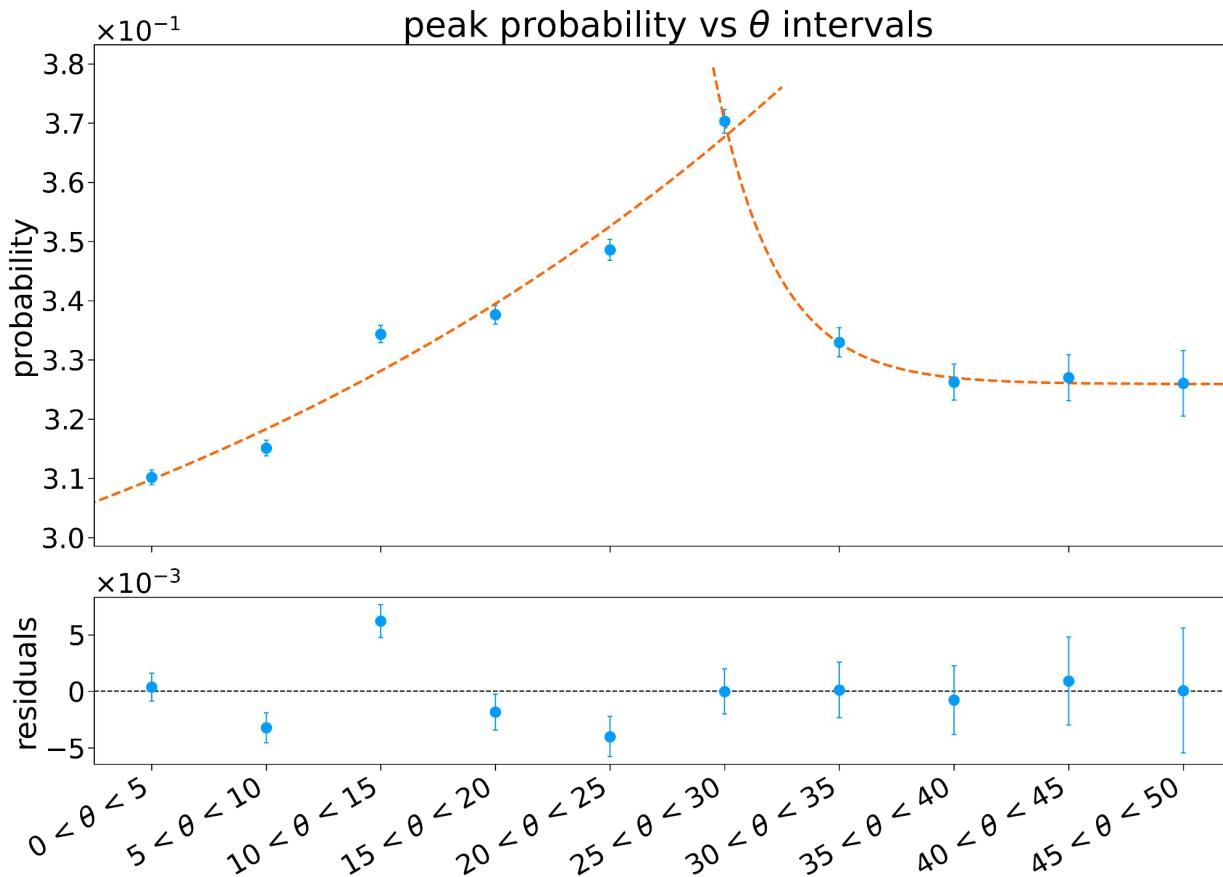


r = 0.8629
p_value = 5.96e-02
chi = 67.51
p_value_chi = 1.456e-14

r = 0.99369
p_value = 6.02e-04
chi = 2.53018
p_value_chi = 0.46986



Density: Middle cut fit

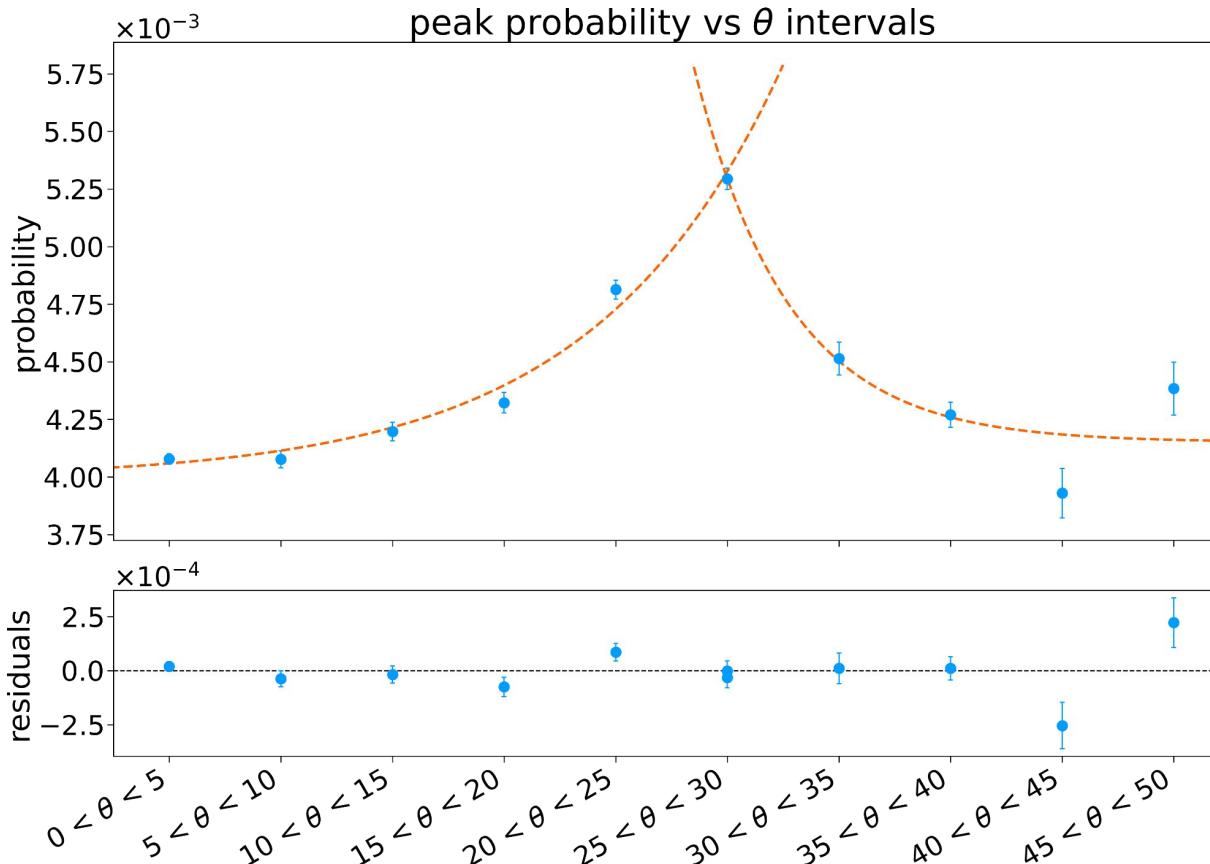


chi = 32.677
p_value_chi = 3.766e-07

chi = 0.1209
p_value_chi = 0.9413



Density: Middle cut gaussian



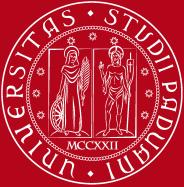
chi = 9.4778
p_value_chi = 0.0207

chi = 9.7582
p_value_chi = 1.45e-12

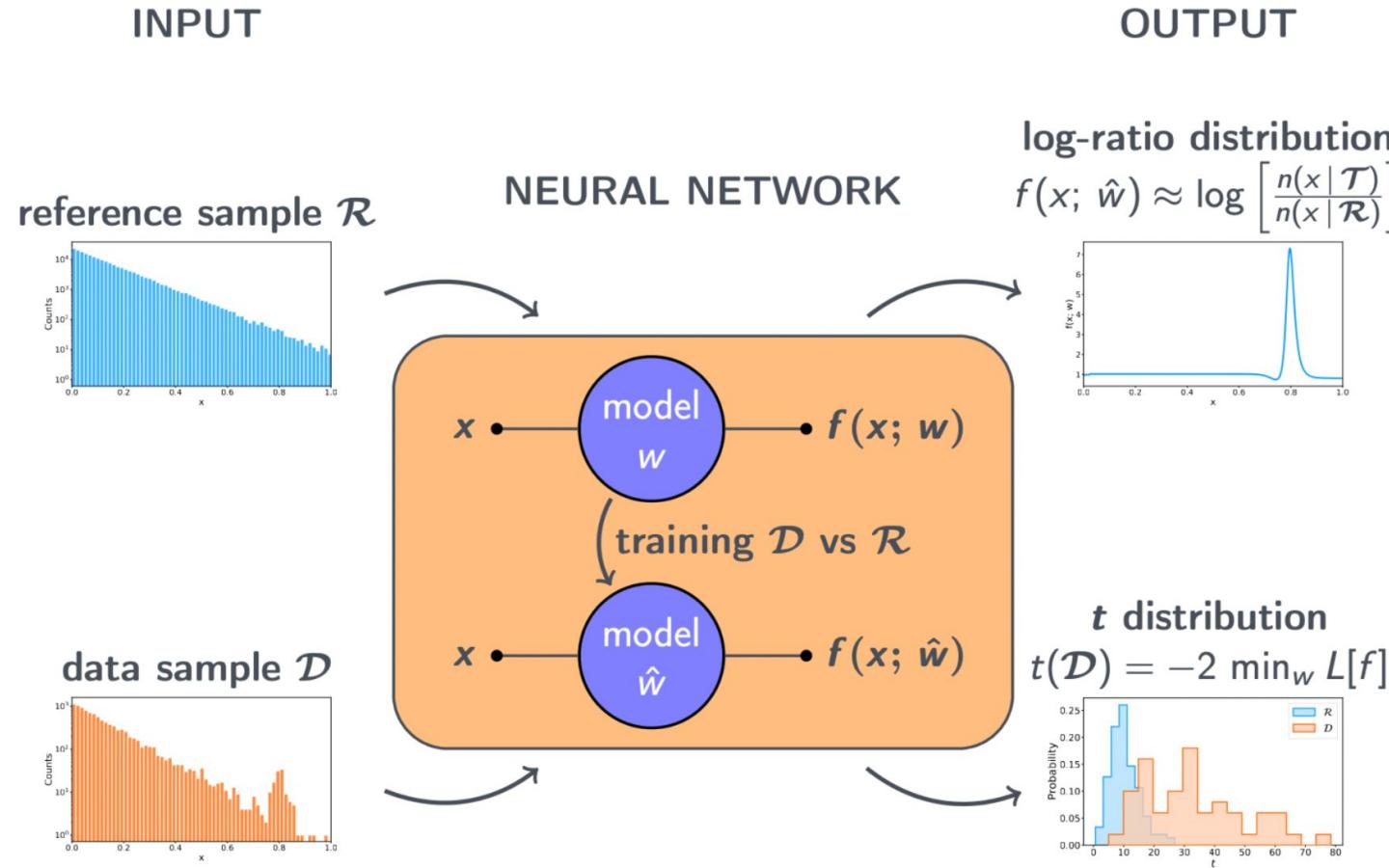


NPLM Algorithm

DQM performance on a 2D input dataset

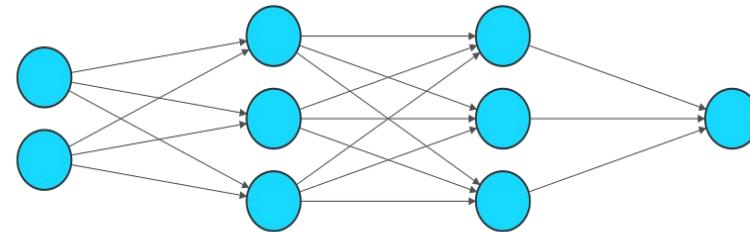


NPLM algorithm reminder





Tuning the Neural Network



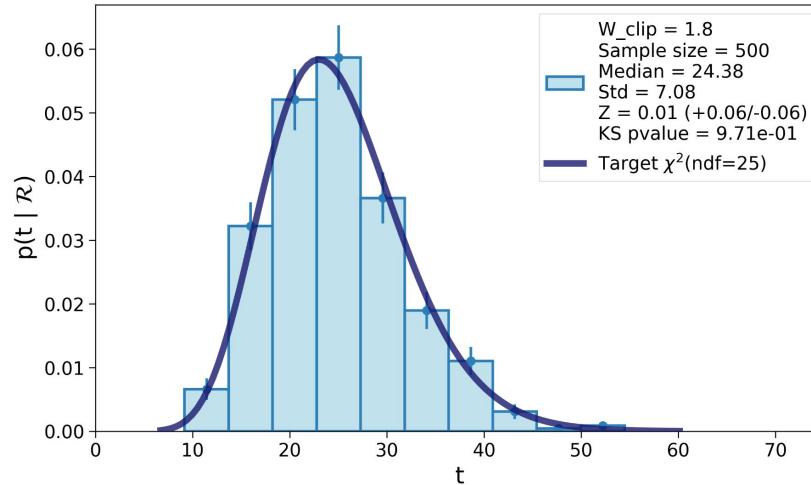
Input Layer $\in \mathbb{R}^2$

Hidden Layer $\in \mathbb{R}^3$

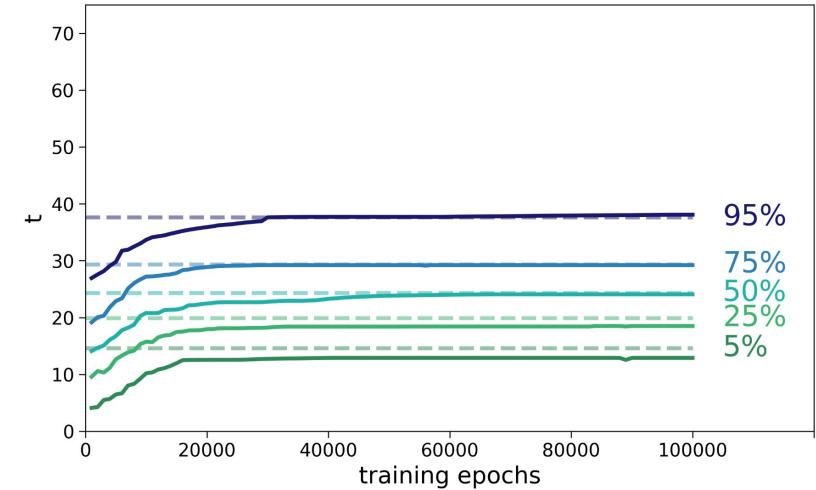
Hidden Layer $\in \mathbb{R}^3$

Output Layer $\in \mathbb{R}^1$

Test statistic distribution

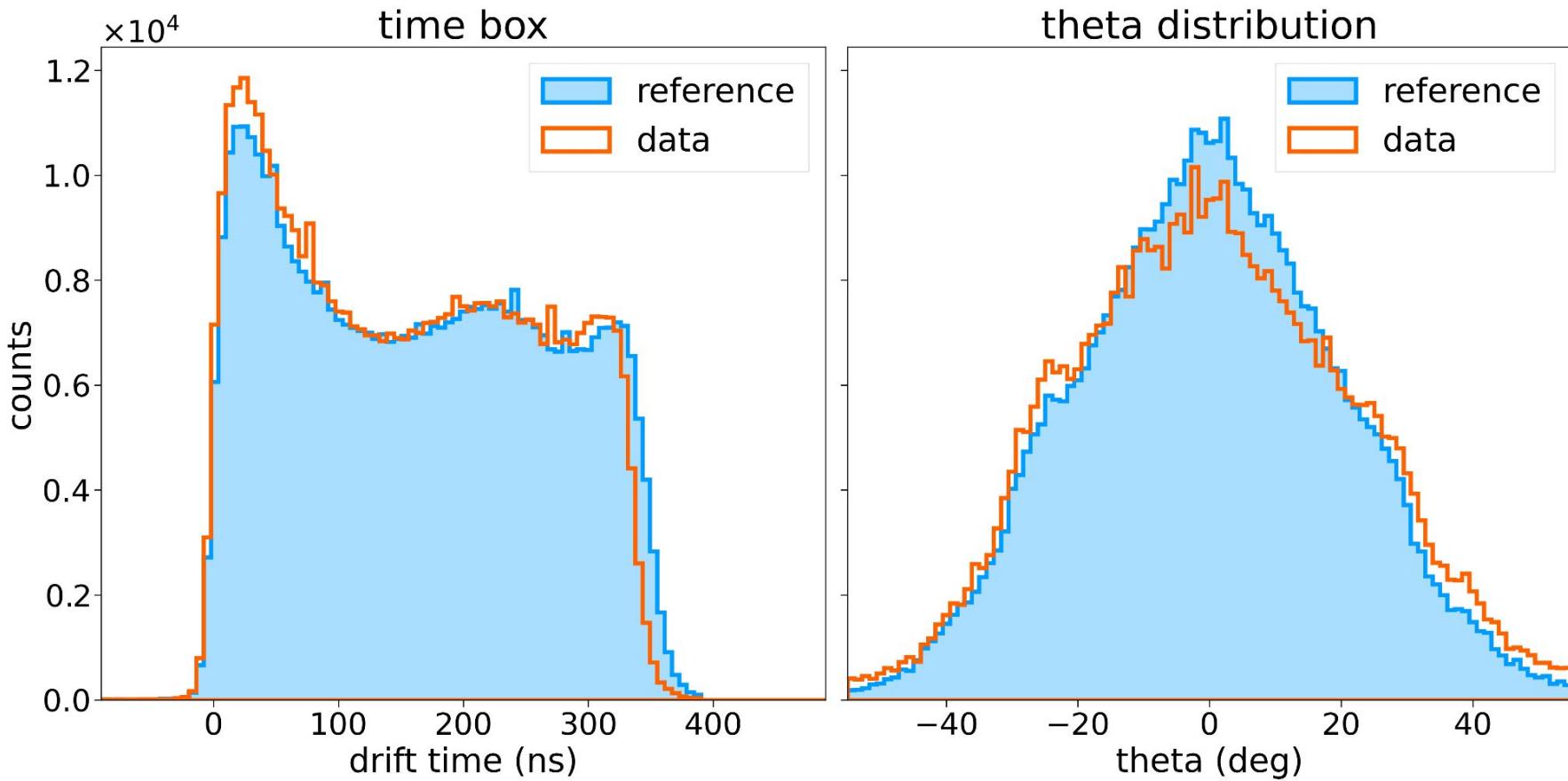


Percentiles evolution





Datasets: Reference and Data





NPLM output

Untouched datasets

Weight clipping = 1.8

- $N_{\text{Reference}} =$

200'000

- $N_{\text{Data}} = 10'000$



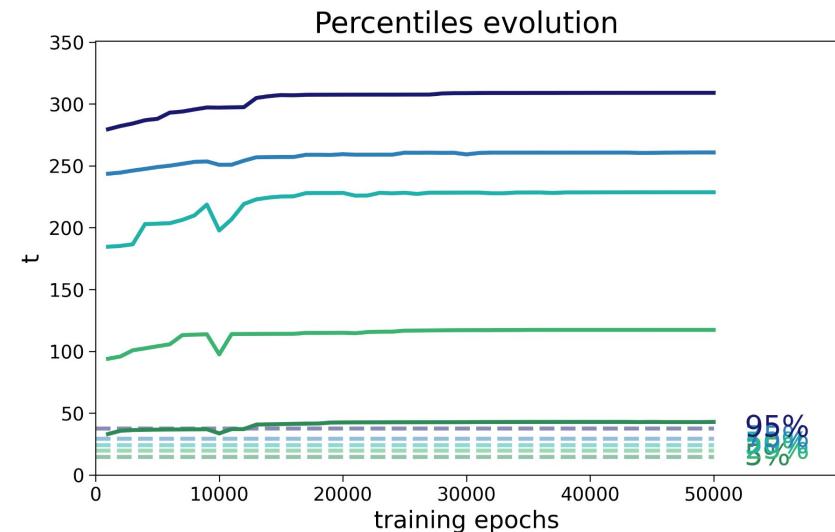
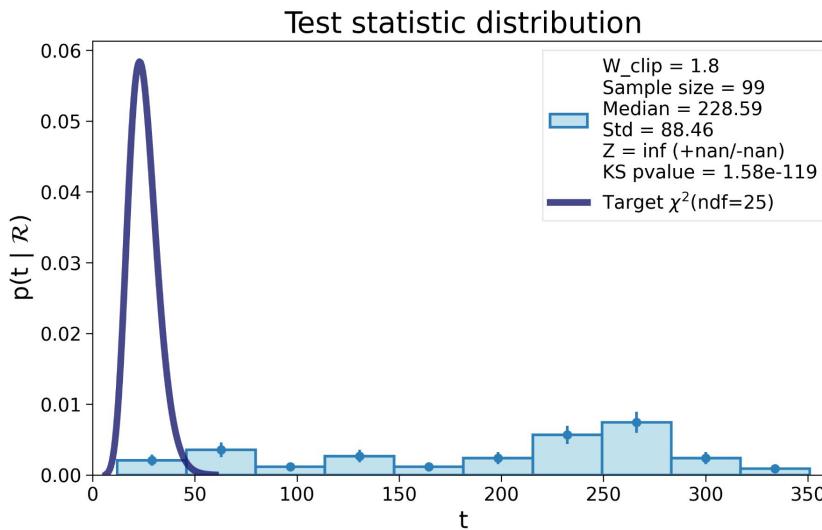
NPLM output

Untouched datasets

Weight clipping = 1.8

• $N_{\text{Reference}} =$

200'000





Cutting the angular feature θ

θ intervals considered:

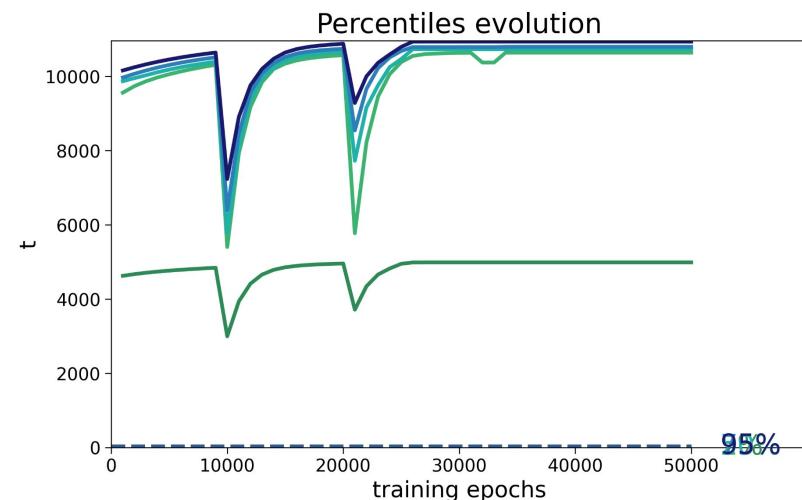
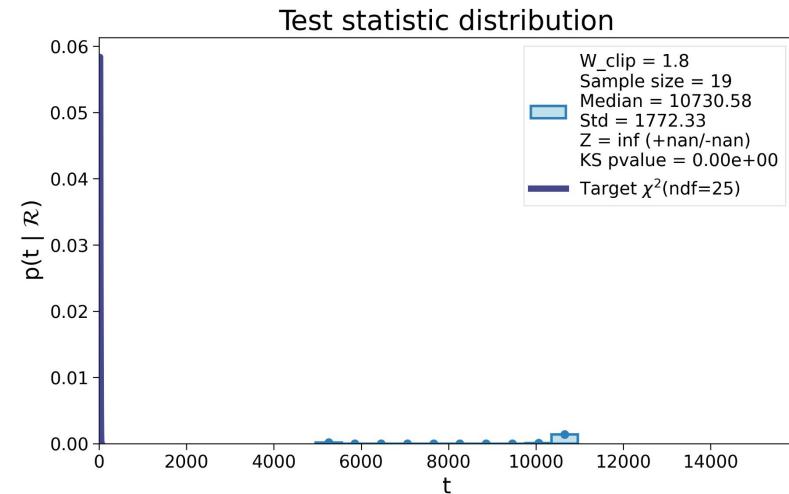
- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$



Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

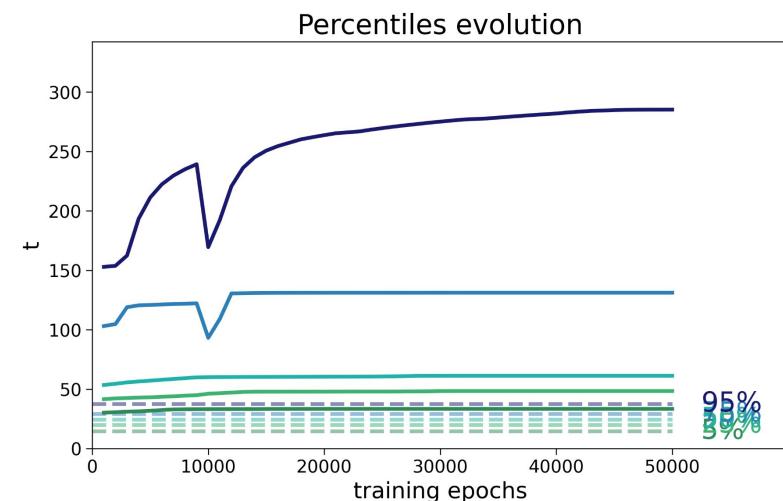
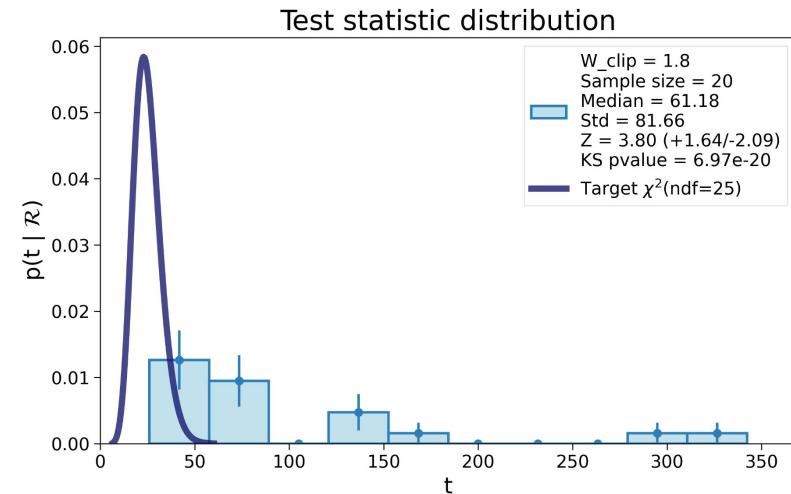




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

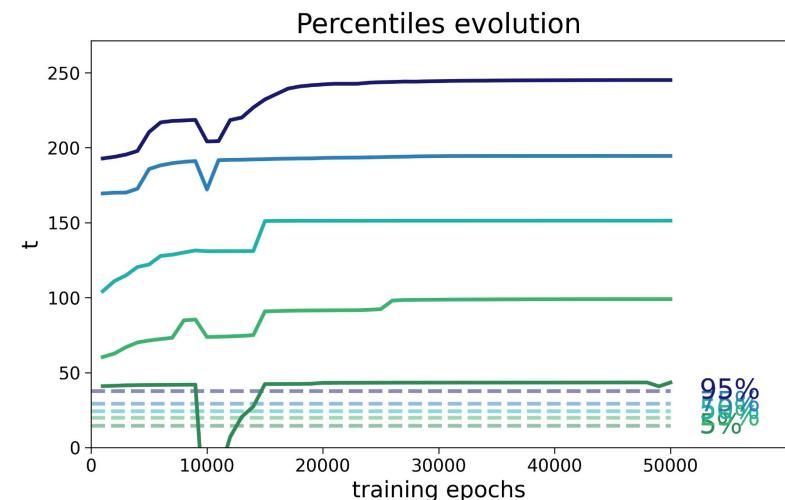
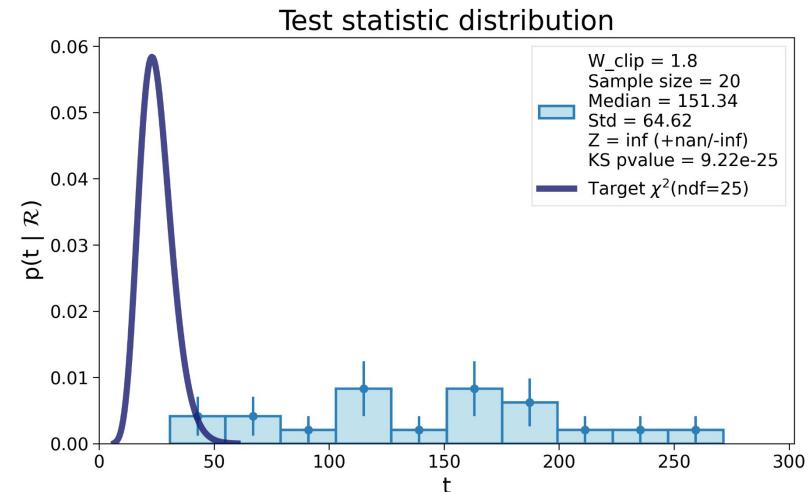




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

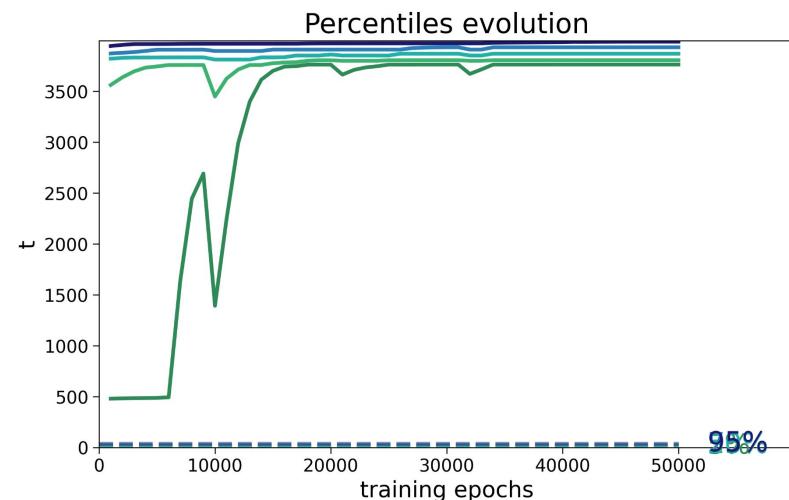
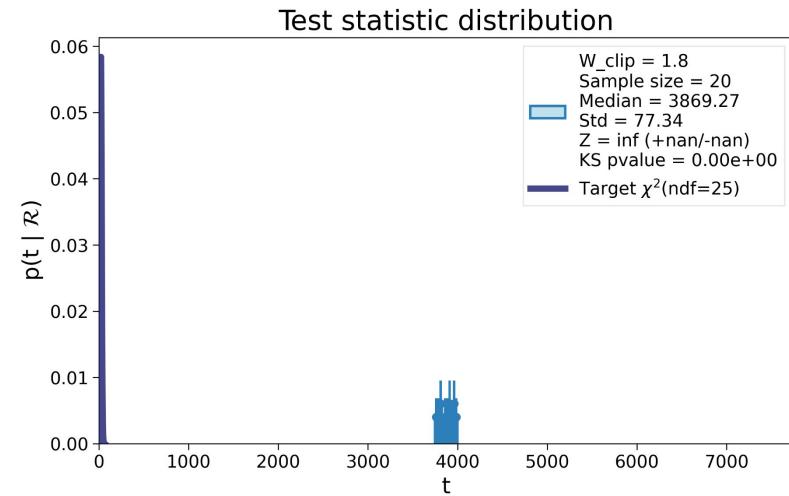




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

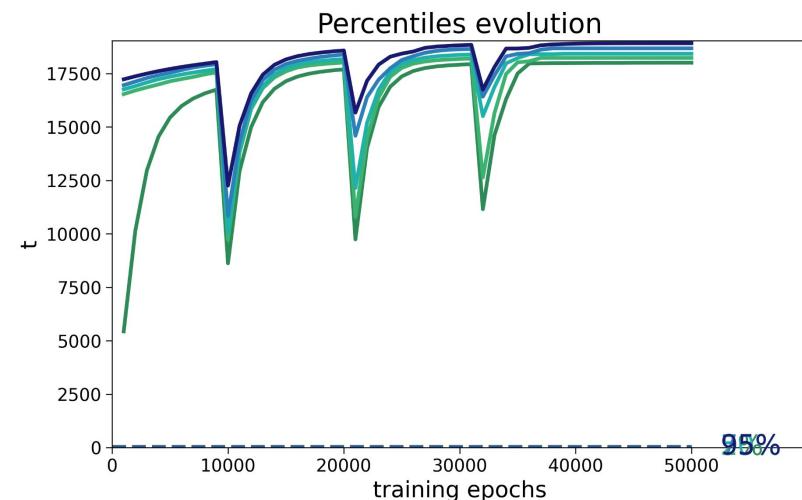
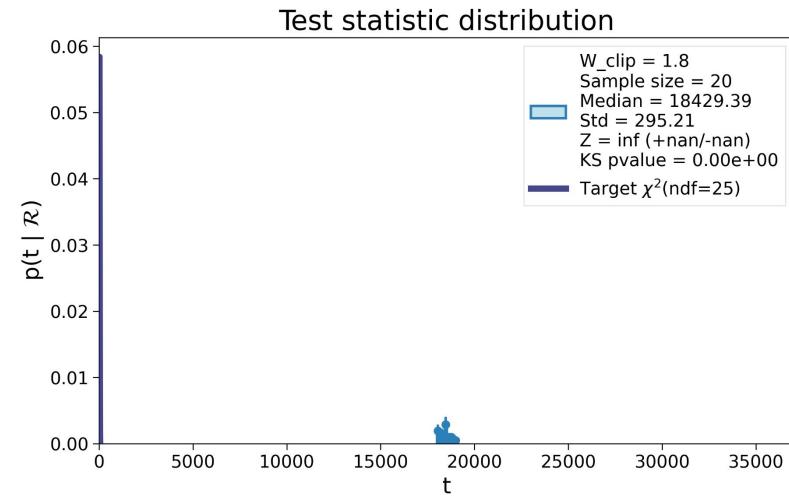




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

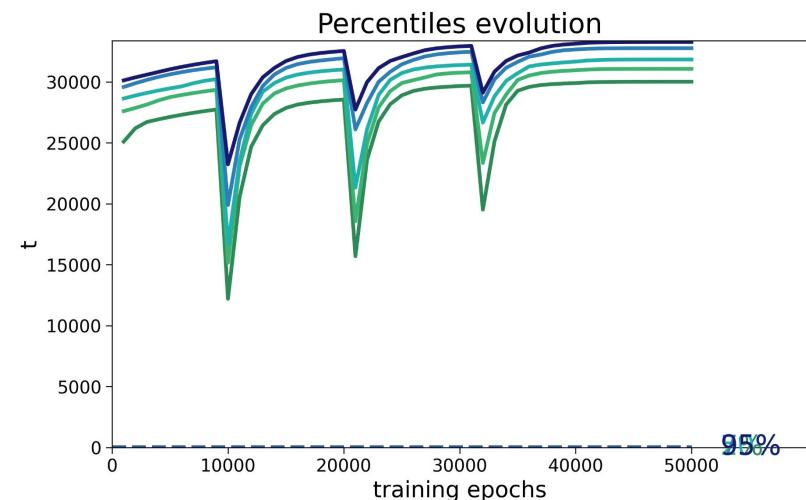
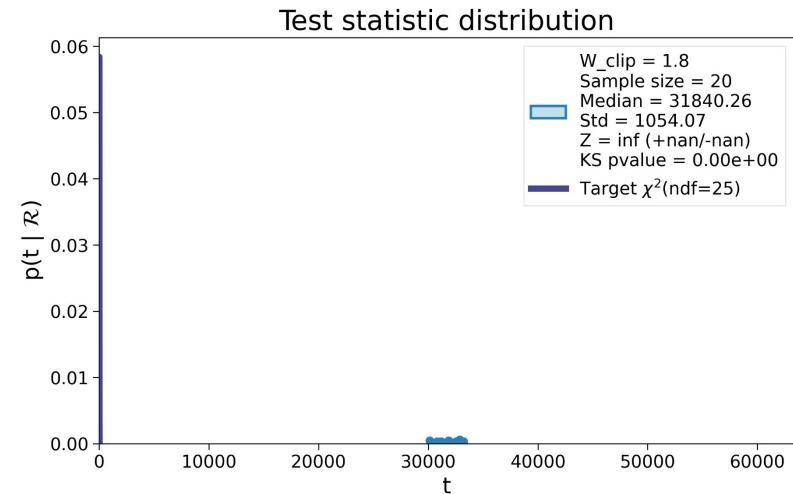


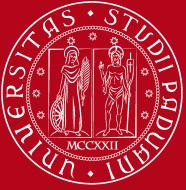


Cutting the angular feature θ

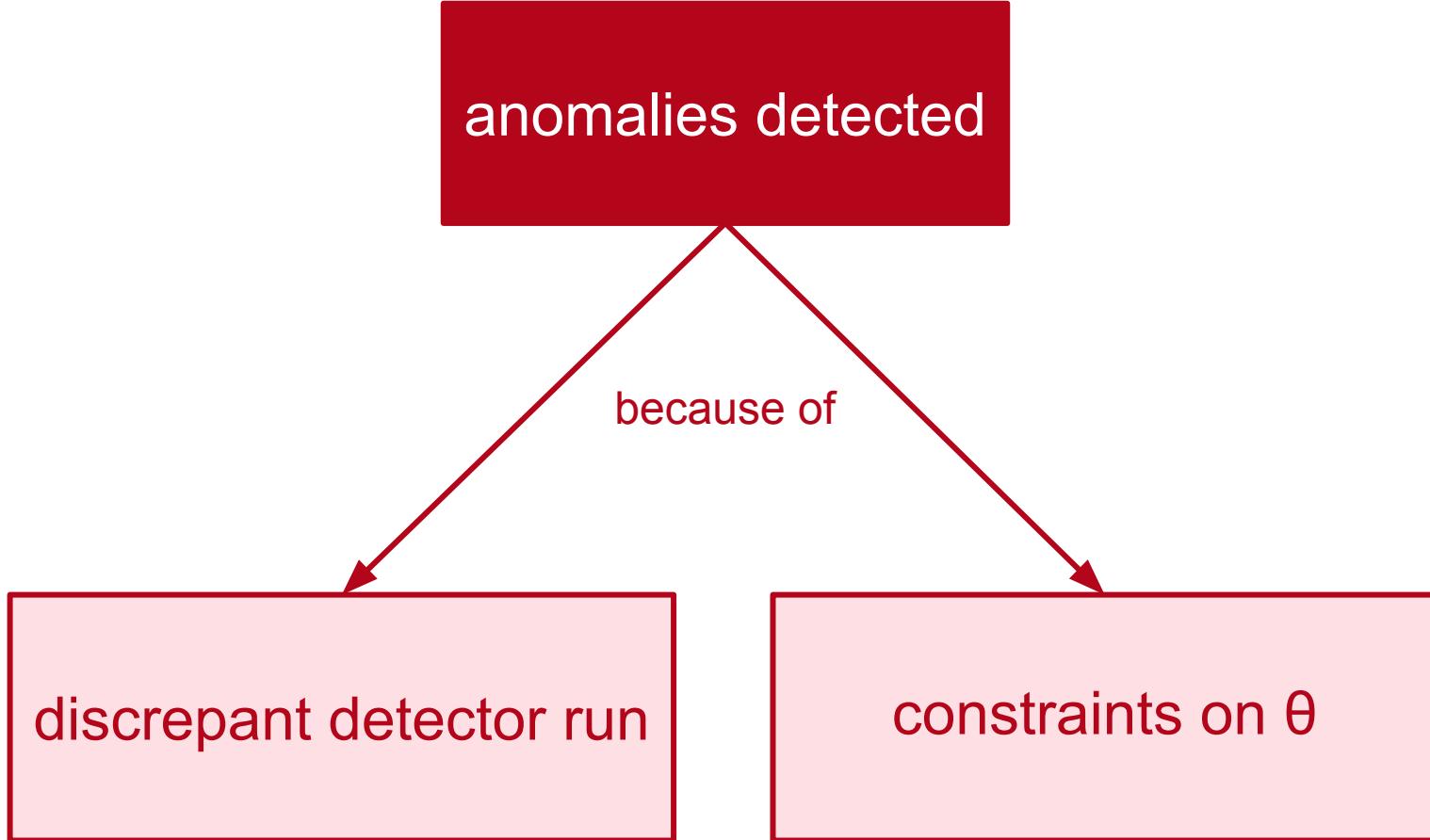
θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$





Cross-check using Reference





Cutting the angular feature θ

θ intervals considered:

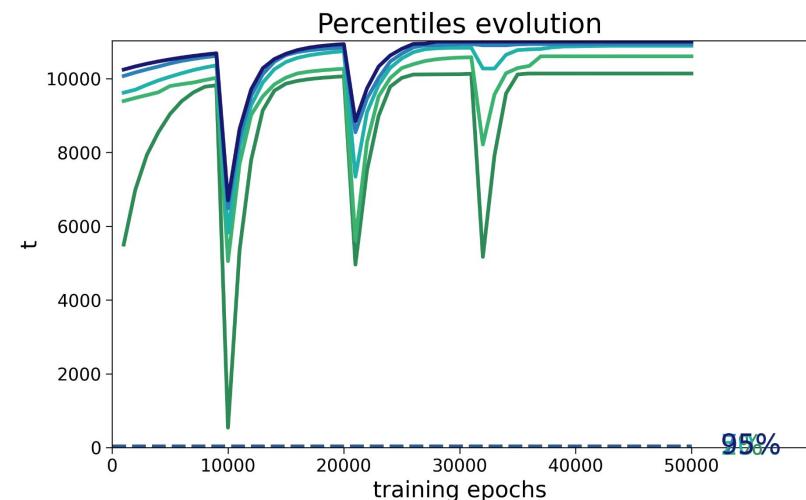
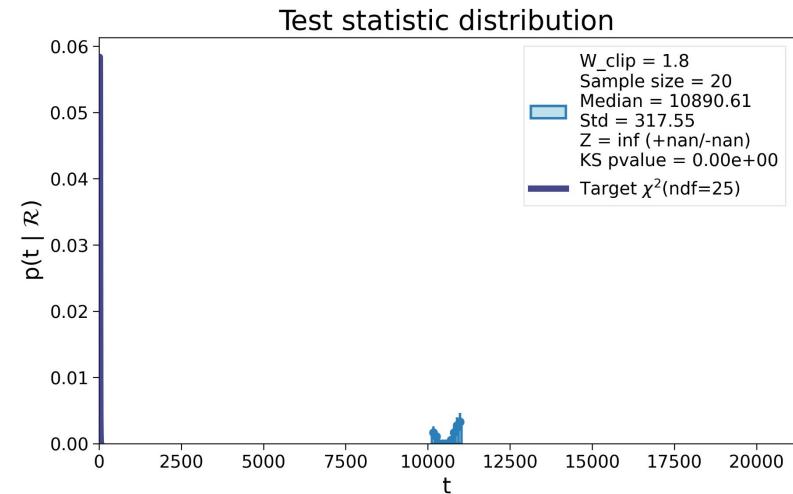
- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$



Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

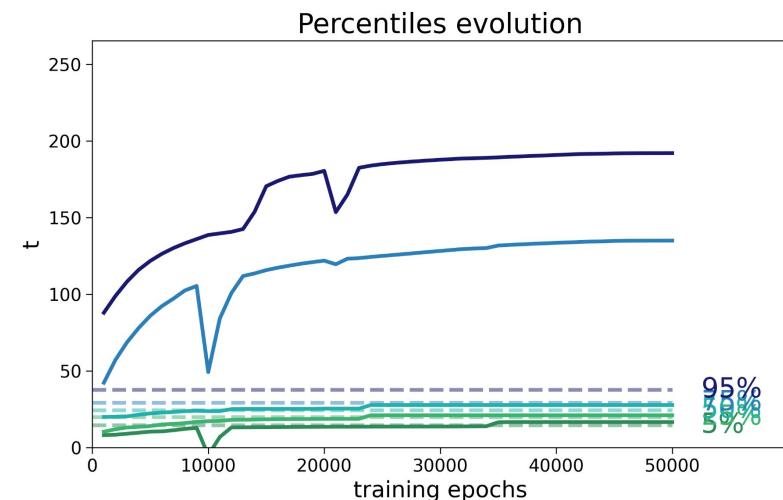
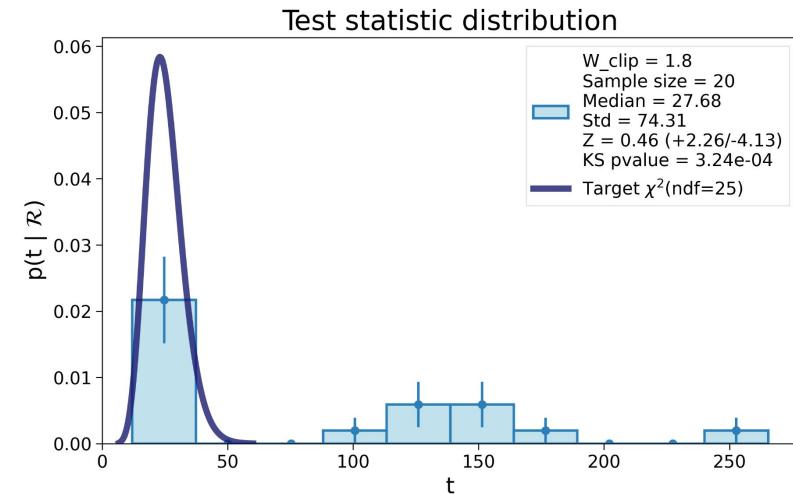




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

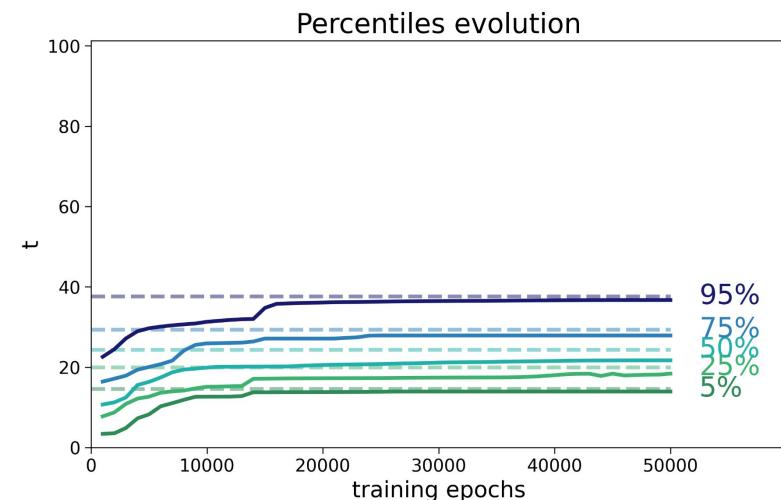
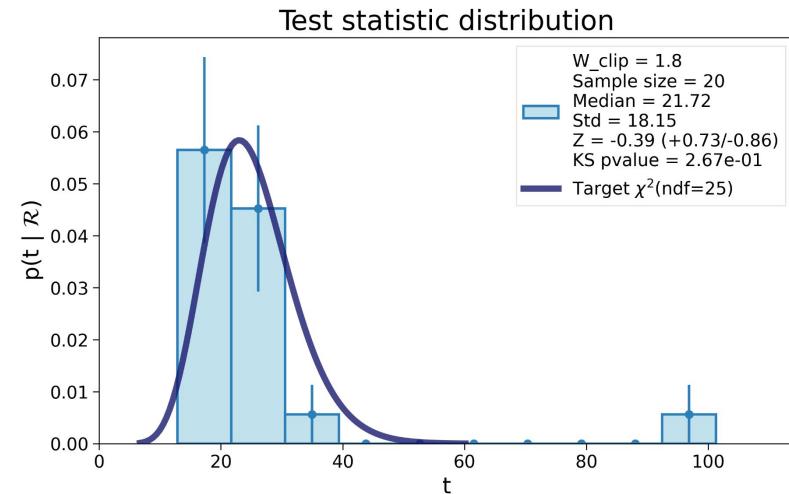




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

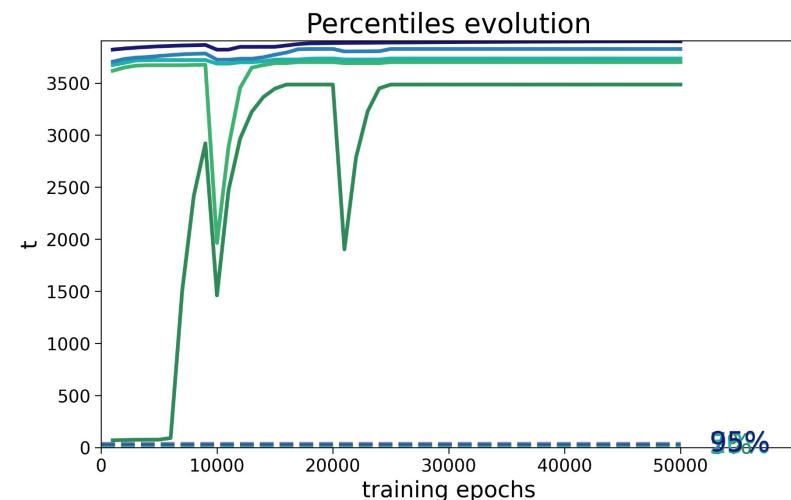
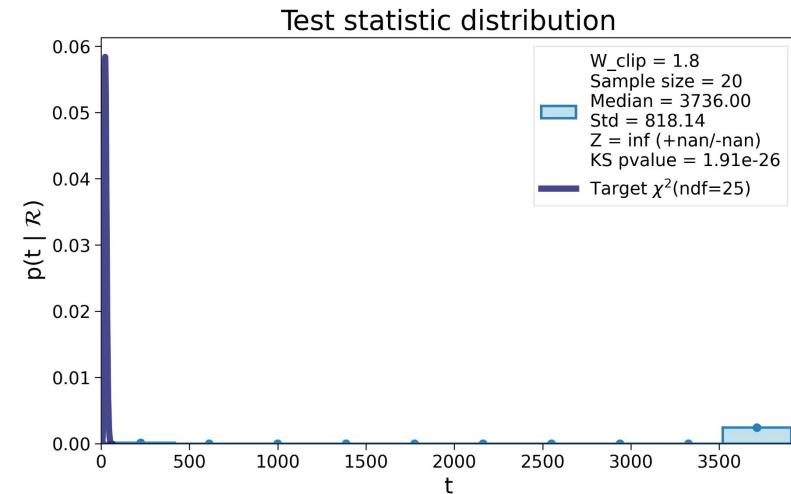




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

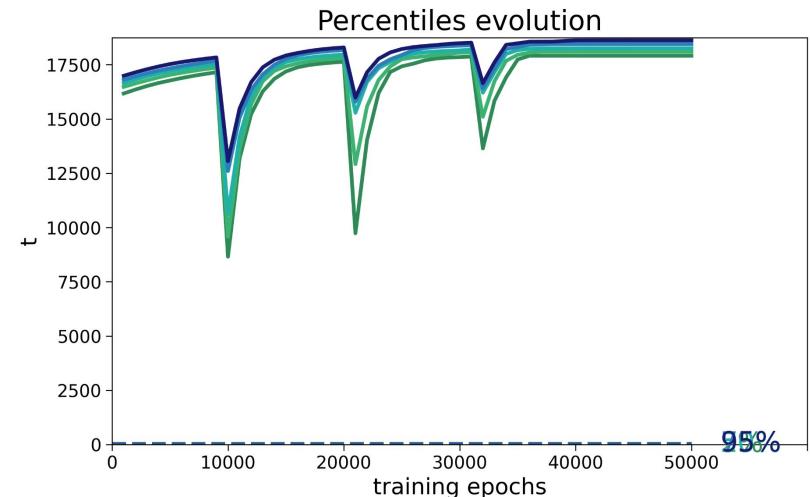
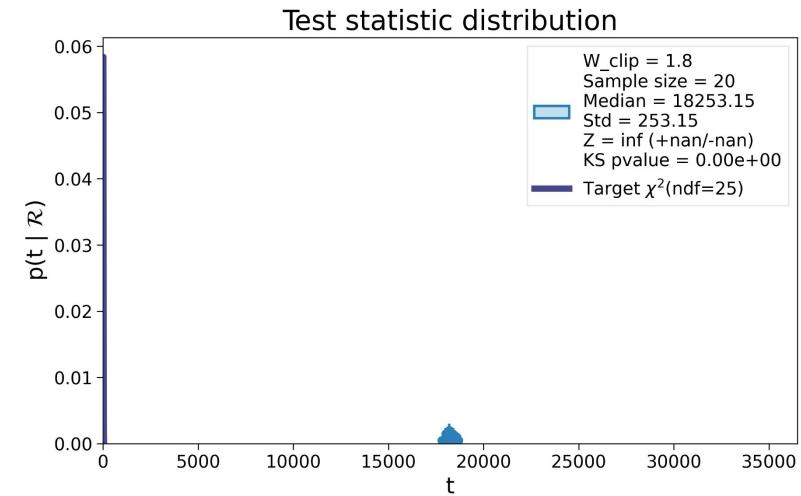




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$

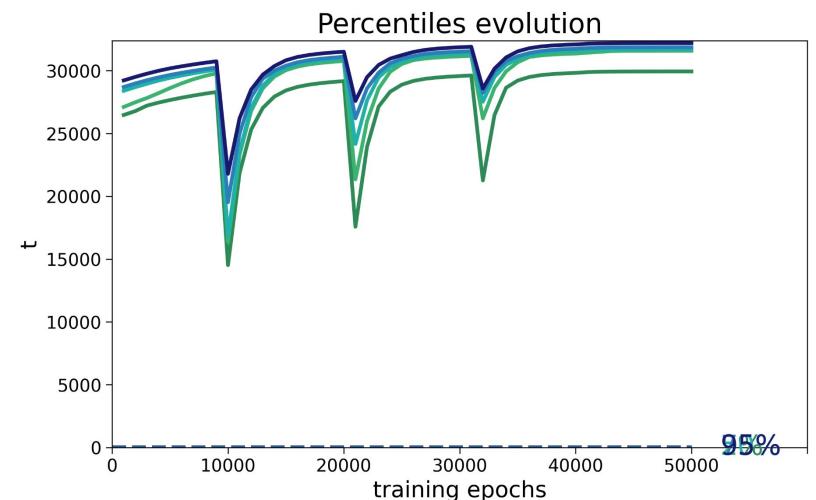
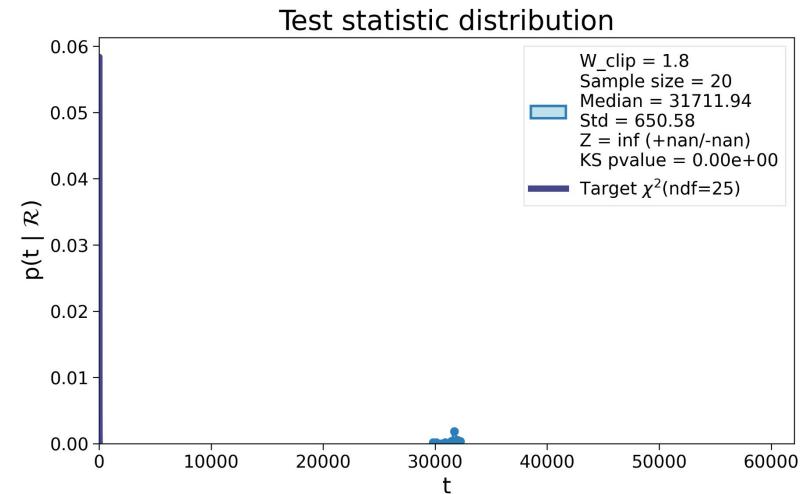




Cutting the angular feature θ

θ intervals considered:

- $0^\circ \leq \theta \leq 15^\circ$
- $0^\circ \leq \theta \leq 45^\circ$
- $0^\circ \leq \theta \leq 50^\circ$
- $5^\circ \leq \theta \leq 45^\circ$
- $20^\circ \leq \theta \leq 40^\circ$
- $30^\circ \leq \theta \leq 55^\circ$





Summary of NPLM results

RUN 57:

- Anomaly detected without θ restrictions
- Anomaly detected with all θ restrictions
 - huge anomalies when
 - no small angles
 - no mid-to-large angles
 - similar to no θ restrictions when
 - no large angles



Summary of NPLM results

RUN 54 (reference):

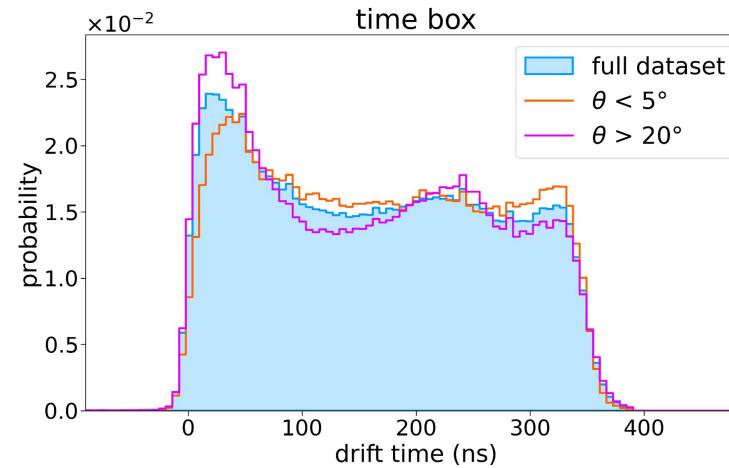
- No anomaly detected without θ restrictions (by construction)
- Anomaly detected with all θ restrictions
 - huge anomalies when
 - no small angles
 - no mid-to-large angles
 - no anomalies with some discrepant test statistic
 - no large angles



False positives and nuisance parameters

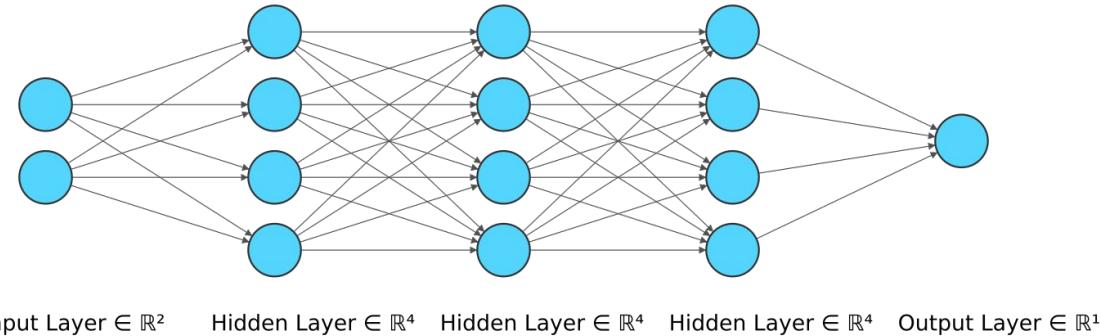
Type I errors in DQM:
satisfactory run classified as discrepant

Example:
time box shape correlated with θ restrictions

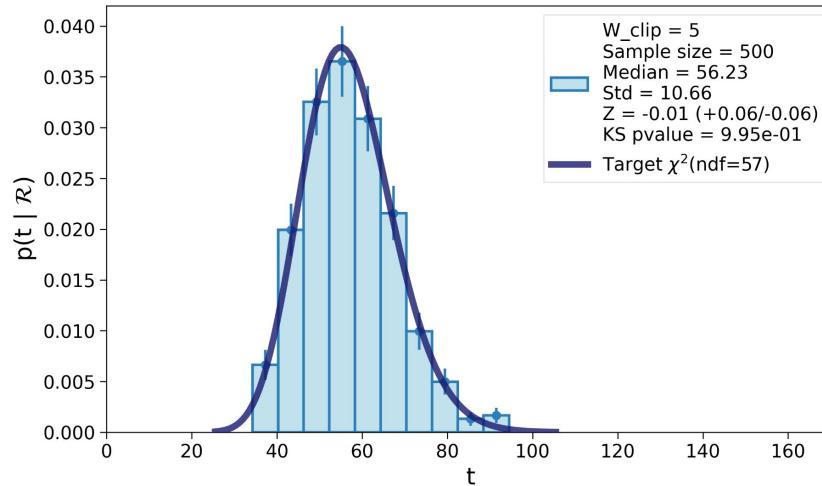




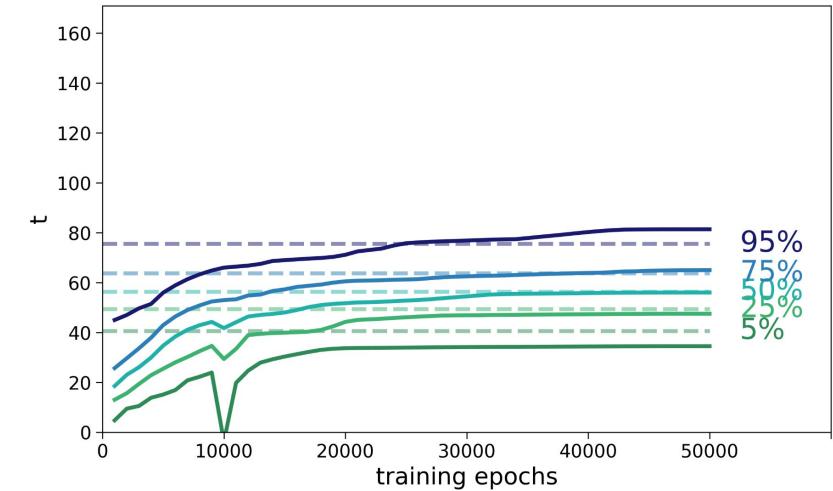
More complex architecture test



Test statistic distribution



Percentiles evolution





NPLM output

Untouched datasets

- $N_{\text{Reference}} =$

Weight clipping = 5

200'000

- $N_{\text{Data}} = 5'000$



NPLM output

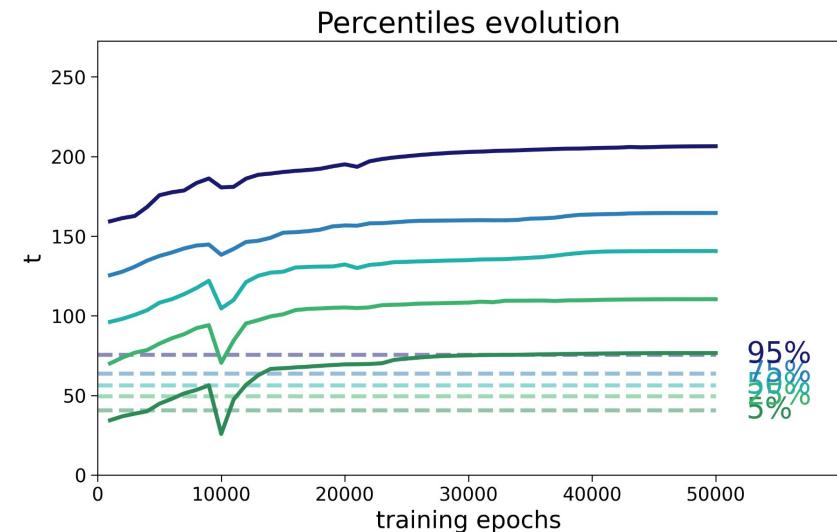
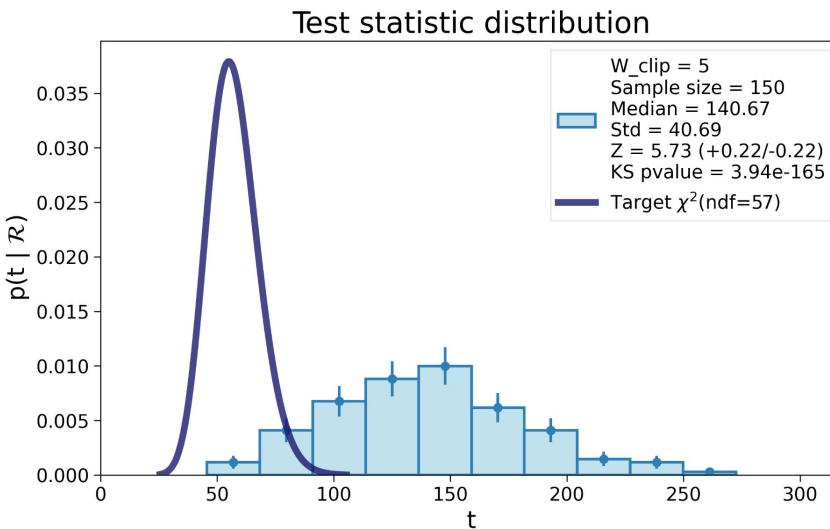
Untouched datasets

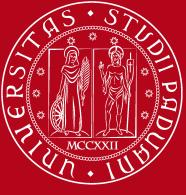
Weight clipping = 5

- $N_{\text{Reference}} =$

200'000

• $N = 5'000$





Conclusions



Summary

- Muon track reconstruction
- Memory and run time optimization
- Search for correlations
- NPLM application and tests on 2D datasets



Thank you
for your attention