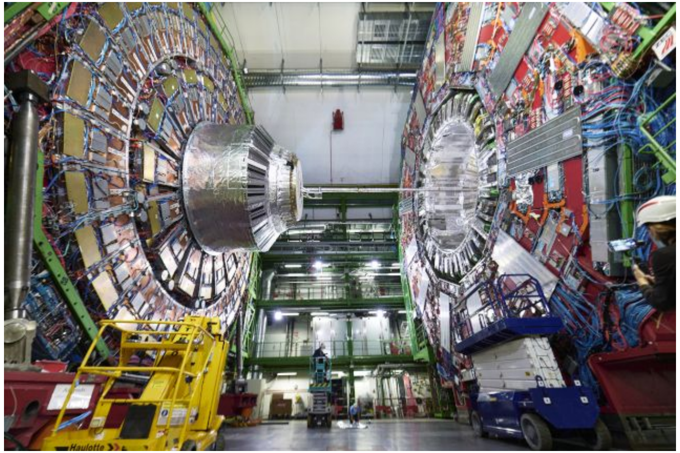
SmartPixels: *Predict track parameters and uncertainties*

Speaker: Arghya Ranjan Das
Purdue University

Challenges in High-Luminosity Environments

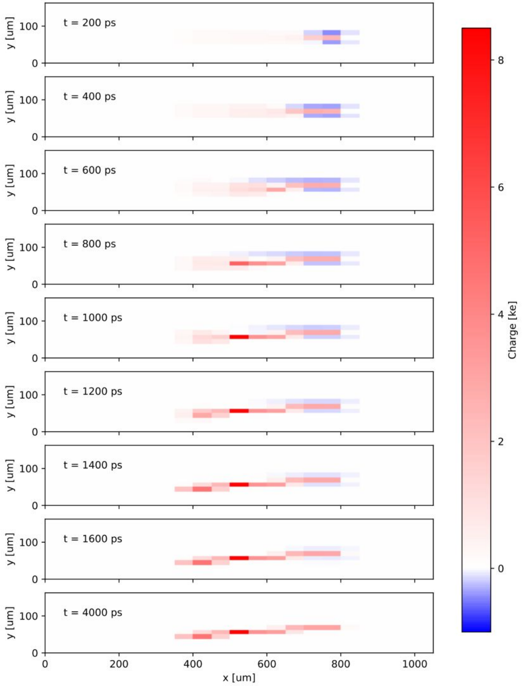
- **Event Rate:** At 40 MHz, the Large Hadron Collider (LHC) generates enormous data volumes, creating a bottleneck in processing and storage.
- **Pixel Detectors' Role**: Pixel detectors have high granularity (future detectors will have even more granularity) which are essential for tracking, vertexing, and flavor tagging.
- **Data Constraints:**
  - Future detectors with high precision will produce extensive amount of data.
  - With Higher Luminosity we have more hits but also backgrounds
  - Low-level triggers prioritize other subsystems, leading to missed events if only pixel data indicate new physics, as in certain Beyond Standard Model (**BSM**) scenarios.



Build you own CMS detector from here

Proposed Solution: On-Sensor Machine Learning (ML)

- **Objective**: Extraction of important pixel information for high-priority physics events by making data-driven decisions.
- **Approach**: Implement a compact, low-latency neural network (NN) directly on the sensor.
- **Challenges with on-chip ML: Low Latency Requirements**
    - The NN must have low latency.
    - Use highly quantized data (2-4 bits) to store charge cluster information.
    - The NN should have small number of parameters.
    - For data reduction process only the first and last time slices out of 20 time slices.
    - The NN should be be optimized further by pruning redundant weights.



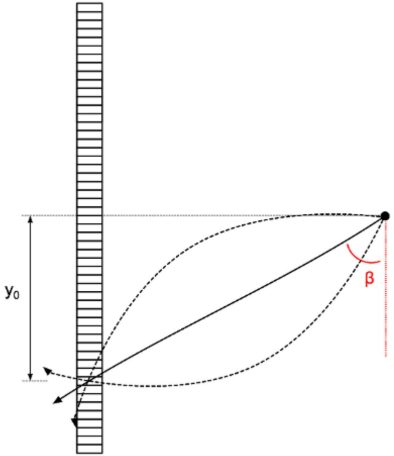
Variable of interest

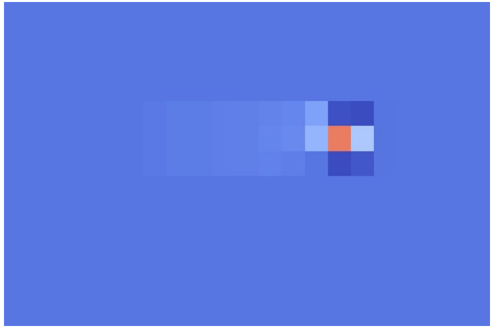
- Predict the beta (β) angle of the particle's trajectory.
- Use the predicted beta angle to estimate the particle's transverse momentum (PT).

$$\beta = \pi/2 - \Delta\phi - \arctan(y_0/R)$$

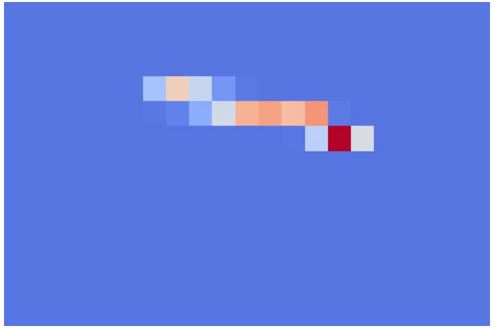
$$\sin(\Delta\phi) = qRB/(2p_T)$$

- x and y coordinates: To locate the position of the particle
- Also predict the alpha (α) angle, it can have some dependency on the beta (β) angle.





First time step



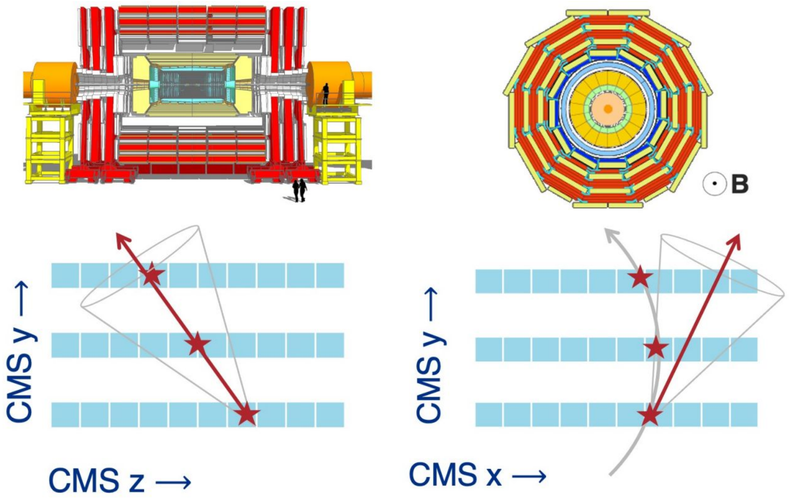
Last time step

Angles and uncertainty

By predicting the angle along with its uncertainty, we can define a cone of expected hit locations in the next detector layer. This helps to reduce combinatorial complexity:

- Smaller uncertainty results in a narrower cone, which further reduces potential hit combinations.

Hence , enables faster tracking and vertexing

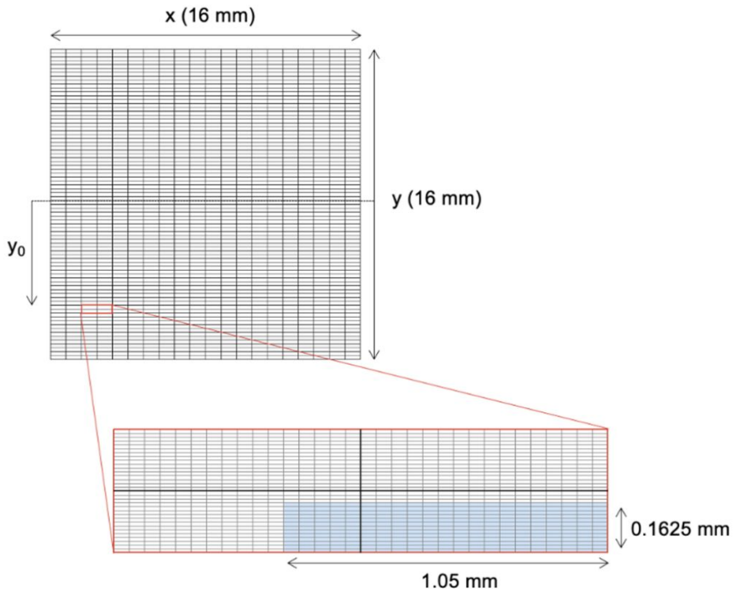


Pixel Geometry and Dataset

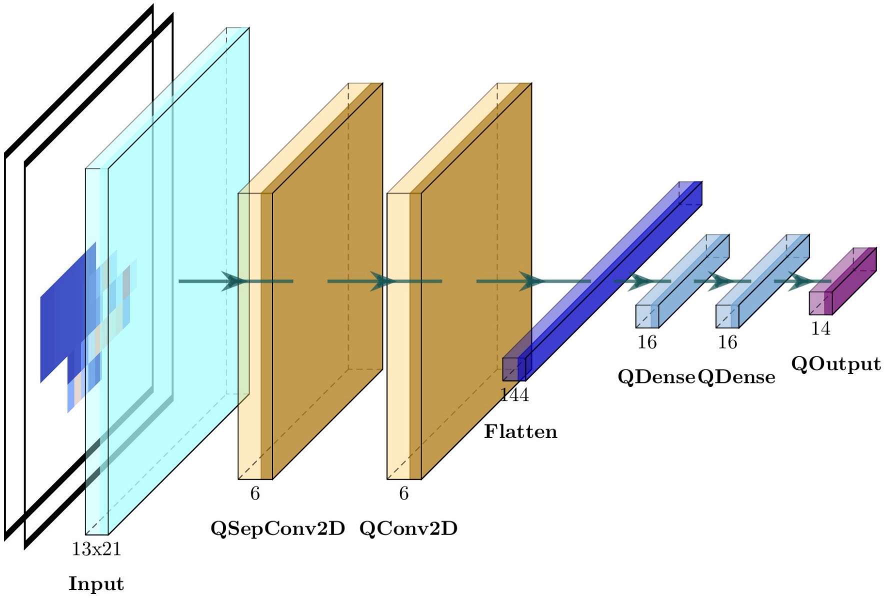
Assuming futuristic pixel geometry:

- 21x13 array of pixels
- 50x12.5 μm pitch, 100 μm thickness
- Located at radius of 30 mm
- 3.8 T magnetic field
- Time steps of 200 picoseconds (T(t=20)=4ns)

Dataset Link: here



Model Architecture (MDN)



SmartPix Model Diagram

Network Type: Mixture Density Network (**MDN**)

Loss function: Negative log-likelihood

Total number of parameters: *2,029*

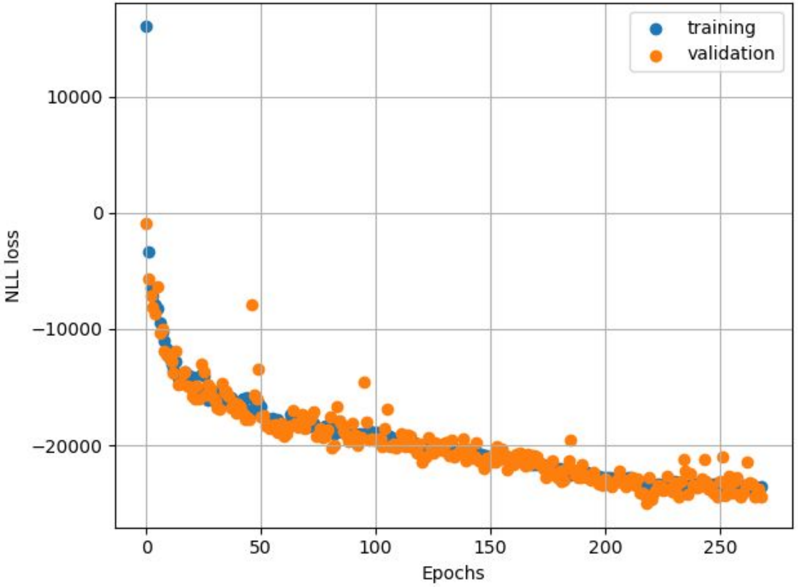
Model Quantization: 8/4 bits

Data Quantization: 4 bits

Output (14 variables)

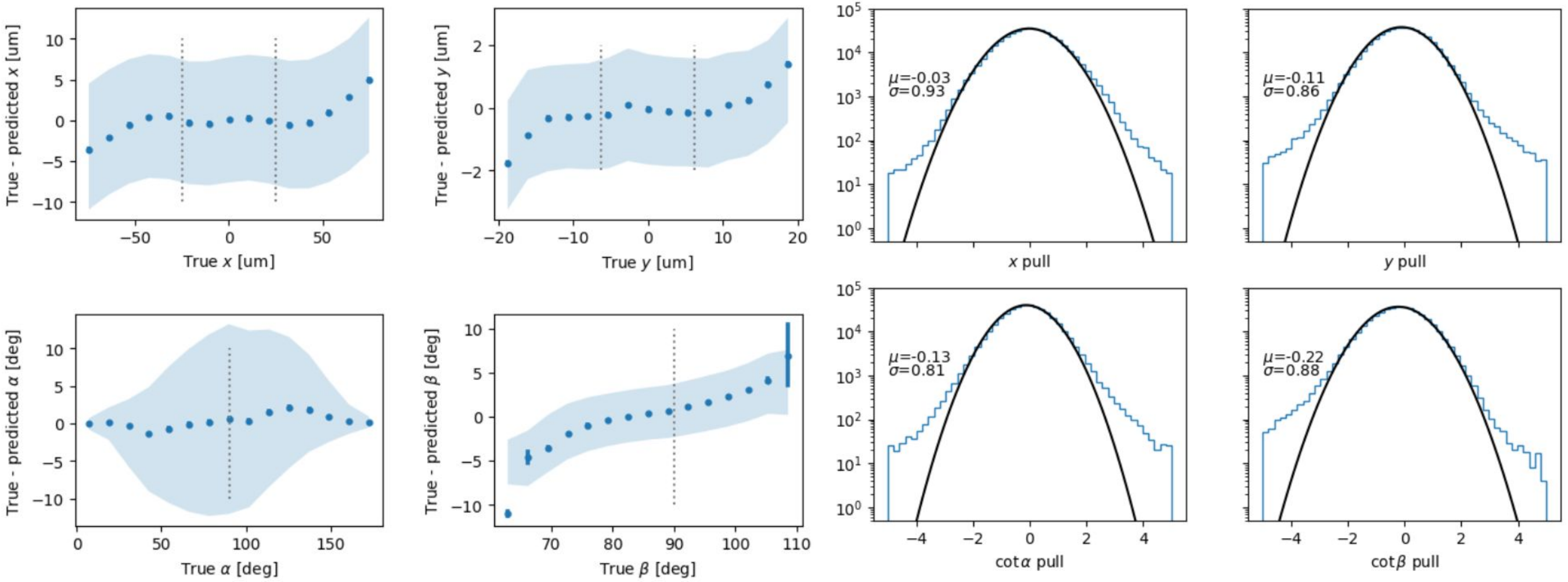
- **4 target variables:** local x, local y, cot α, cot β
- **10 Co-variances:** Measure of uncertainties

Model convergence




```
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 13, 21, 2)]       0

q_separable_conv2d (QSepara  (None, 11, 19, 5)        33
bleConv2D)

q_activation (QActivation)  (None, 11, 19, 5)         0

q_conv2d (QConv2D)          (None, 11, 19, 5)         30

q_activation_1 (QActivation  (None, 11, 19, 5)        0
)

average_pooling2d (AverageP  (None, 3, 6, 5)          0
ooling2D)

q_activation_2 (QActivation  (None, 3, 6, 5)          0
)

flatten (Flatten)           (None, 90)                0

q_dense (QDense)            (None, 16)                1456

q_activation_3 (QActivation  (None, 16)               0
)

q_dense_1 (QDense)          (None, 16)                272

q_activation_4 (QActivation  (None, 16)               0
)

q_dense_2 (QDense)          (None, 14)                238

=================================================================
Total params: 2,029
Trainable params: 2,029
Non-trainable params: 0
```

Current Results

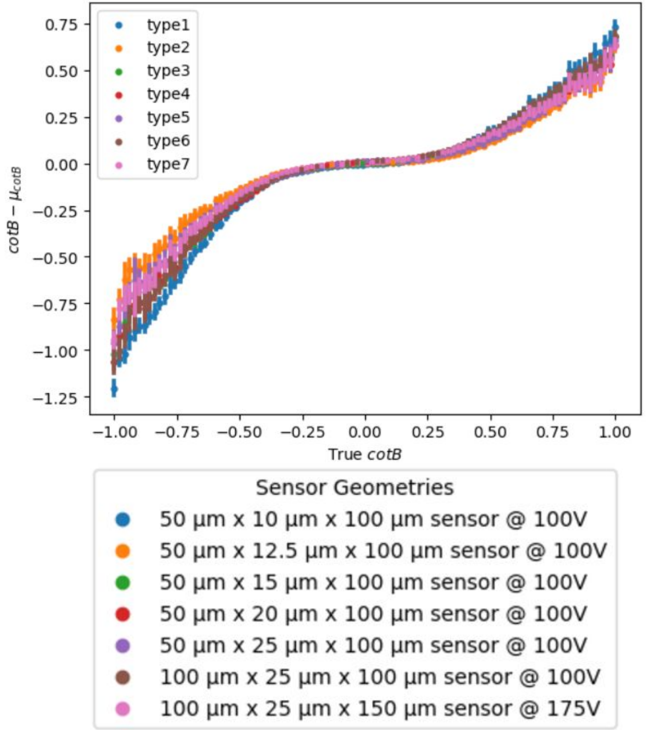


Different Sensor Geometry (dataset 2s)

- Training conducted on all geometries
- Example shown: Comparison for **beta** (see right)
- Visuals become cluttered with all geometries included
- To enhance visual clarity, selected only **two** types of geometries for comparison
- Thus we choose the geometries with same pitches but **different thickness**:

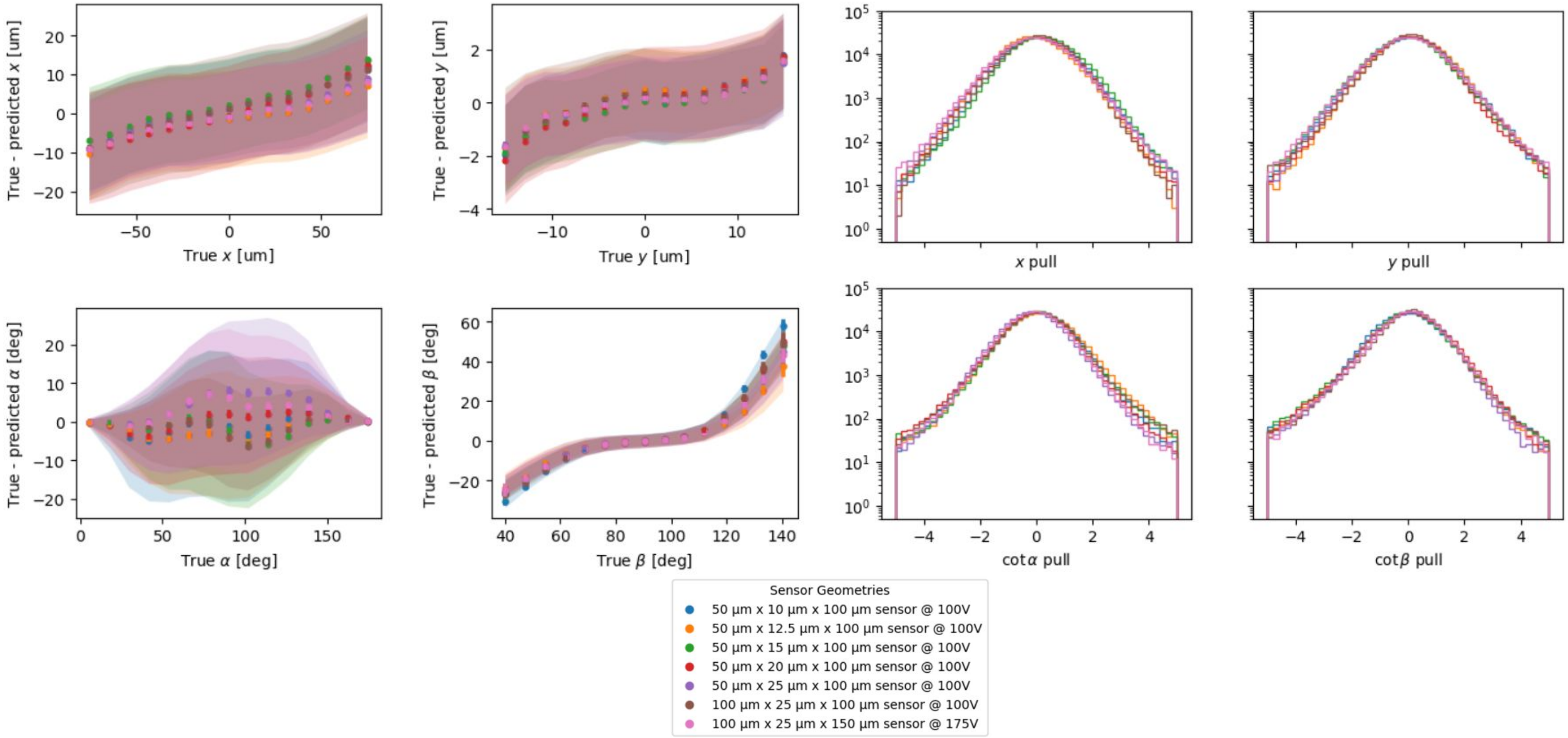
  - (100,25,150)um-175V
  - (100,25,100)um-100V

  And we also study the **dependence** on the number of **time-slices.**

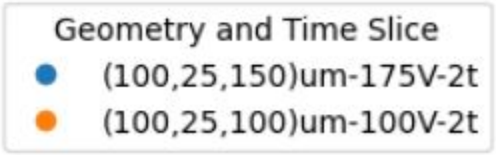


All Trainings are done with batch_size = 5000

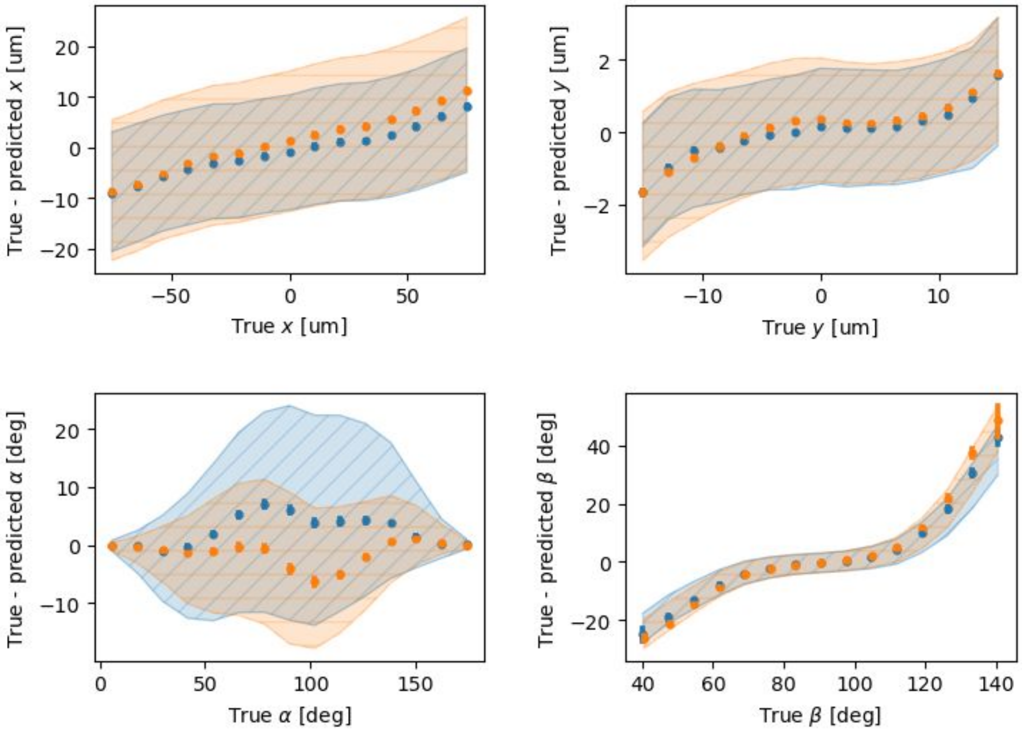
Different Sensor Geometry Results



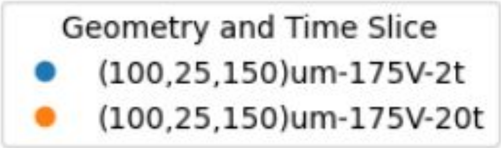
2 Timeslices with different thickness



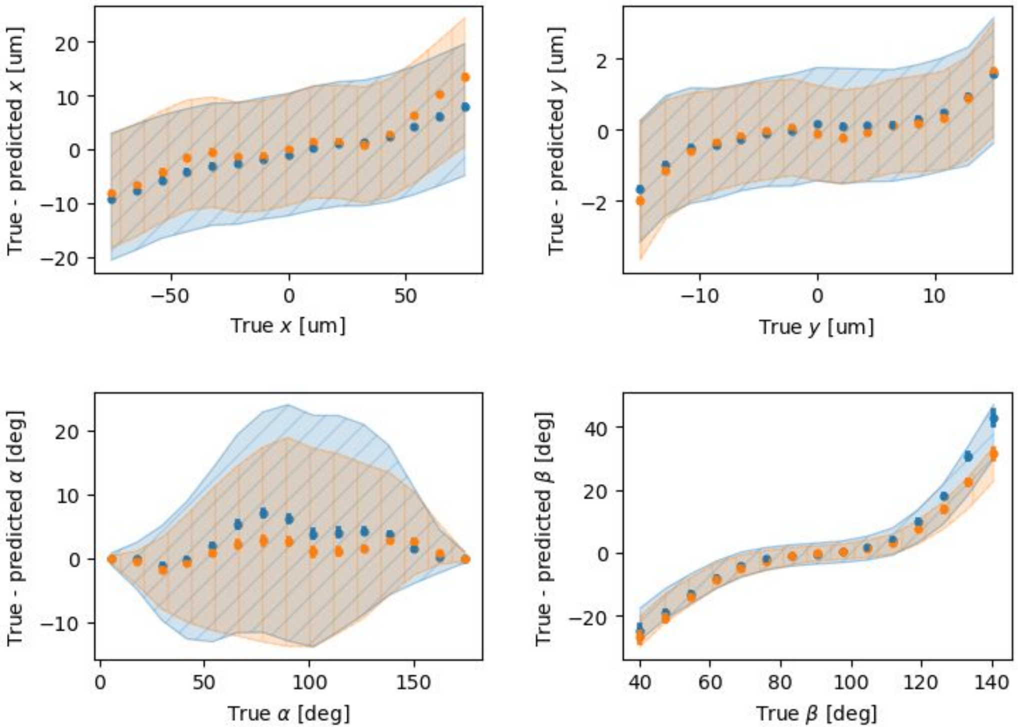
- Intuitively the higher thickness should gives better accuracy.
- And as evident from the plots we are getting better accuracy for higher thickness (except for alpha).



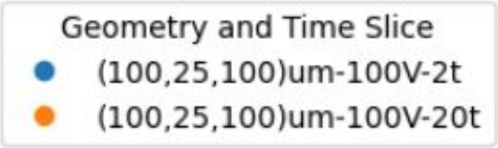
2 vs 20 timeslices



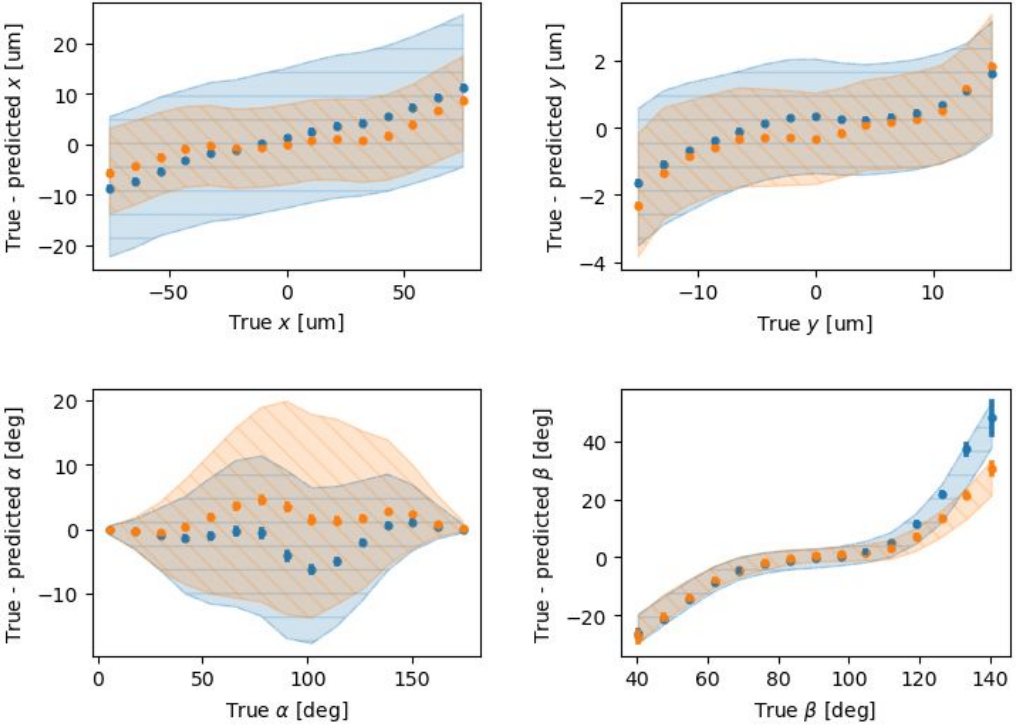
- Used 2 time-slices for model training (full 20 may not be available)
- Observed performance **improves with more time-slices** due to increased information
- Found the **performance loss** when training with only 2 time-slices is **small**.



2 vs 20 timeslices



- We see for x, y, and beta the performance increased with more timeslices. Similar to the last slide.
- Except for the alpha angle. Somehow increasing the time-information decreases the accuracy.



Multi-Objective Optimization

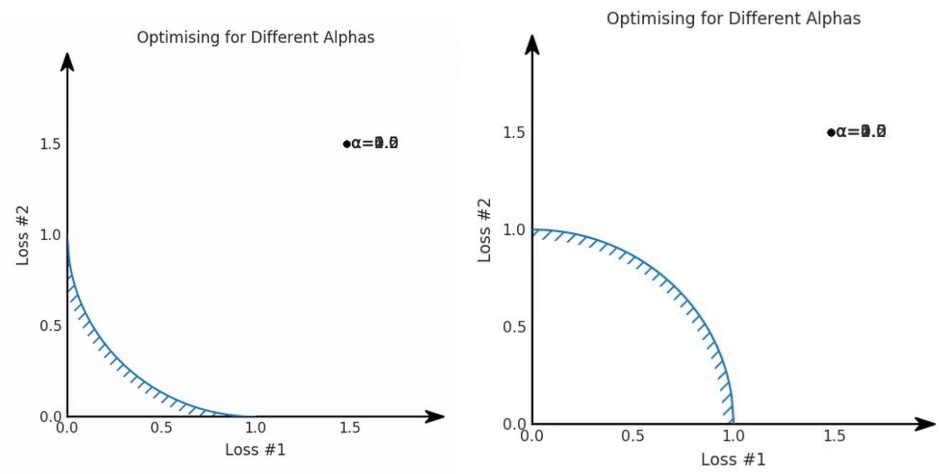
$$Loss = \mathcal{L} = L_0(\boldsymbol{\theta}) + \sum_{\alpha=1}^{N} \lambda_\alpha L_\alpha(\boldsymbol{\theta})$$

Penalty Method (ex: l1/l2 regularizers)

$$\dot{\theta_i} = -\frac{\partial}{\partial \theta_i} L_0(\boldsymbol{\theta}) - \lambda \frac{\partial}{\partial \theta_i} L_1(\boldsymbol{\theta})$$

We want to optimize L0 subject to some constraint L1

- An optimal solution can be determined when the Pareto front is convex, but this does not hold for concave fronts.
- Generally, the Pareto front is unknown and is a mix of both the types.
- Thus there is no mathematical guarantee for optimizing multiple objectives simultaneously



[J. Degrave & I. Korchunov]

Thus we see move to another method of using lagrange multipliers

BDMM (Basic Differential method of Multiplier)

Lagrange Multipliers: Optimal solution found we satisfy the condition

$$\boxed{\frac{\partial \mathcal{L}}{\partial \theta_i} = 0 \quad \frac{\partial \mathcal{L}}{\partial \lambda_i} = 0}$$

Pure gradient descent does not work with Lagrange multipliers due to saddle points. We perform gradient descent on weights and gradient ascent on Lagrange multipliers.

$$\dot{\theta}_i = -\frac{\partial}{\partial \theta_i} L_0(\boldsymbol{\theta}) - \sum_{\alpha=1}^{N} \lambda_\alpha \frac{\partial}{\partial \theta_i} L_\alpha(\boldsymbol{\theta})$$

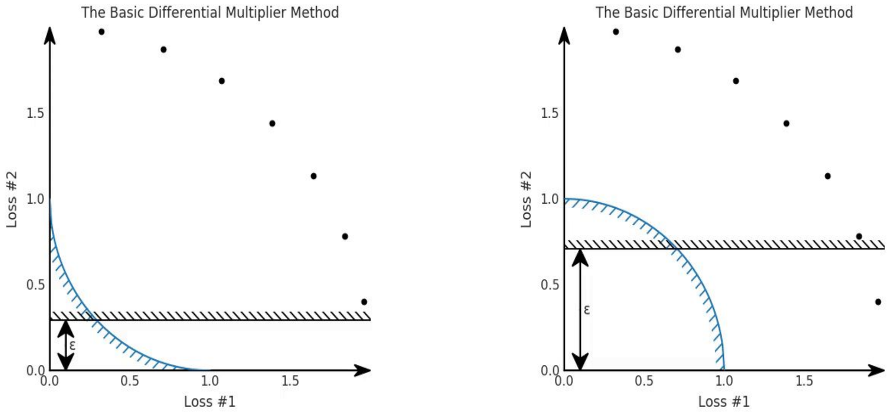
$$\dot{\lambda}_i = L_i(\boldsymbol{\theta})$$

Equation of motion of the weights

$$\ddot{\theta}_i + \sum_{j} \left(\frac{\partial^2 L_0}{\partial x_i \partial x_j} + \sum_{\alpha=1}^{N} \lambda_\alpha \frac{\partial^2 L_\alpha}{\partial x_i \partial x_j}\right) \dot{x}_j + \sum_{\alpha=1}^{N} L_\alpha \frac{\partial L_\alpha}{\partial x_i} = 0$$

$$\dot{E} = -\sum_{i,j} \dot{x}_i \left(\frac{\partial^2 L_0}{\partial x_i \partial x_j} + \sum_{\alpha=1}^{N} \lambda_\alpha \frac{\partial^2 L_\alpha}{\partial x_i \partial x_j}\right) \dot{x}_j = -\sum_{i,j} \dot{x}_i A_{ij} \dot{x}_j$$

We need to add some damping



[J. Degrave & I. Korchunov]

MDMM (Modified Differential method of Multiplier)

We use a technique termed MDMM which is similar to the last method with just and extra penalty term so as to facilitate damping

$$Loss = \mathcal{L} = L_0(\boldsymbol{\theta}) + \sum_{\alpha=1}^{N} \lambda_\alpha L_\alpha(\boldsymbol{\theta})$$

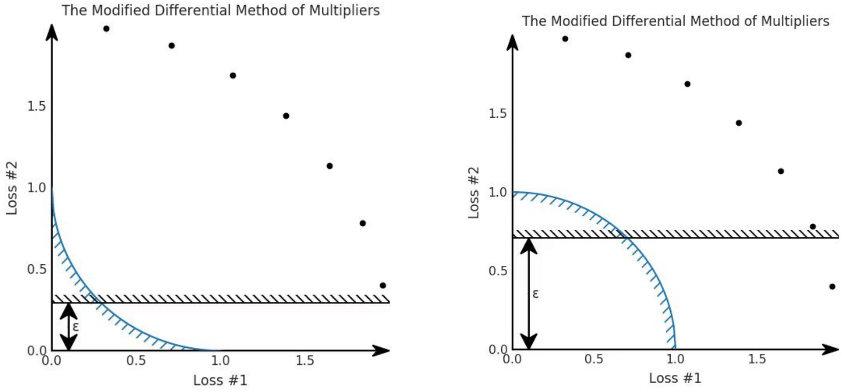
Algorithm: Define the loss as (everything else stays the same)

$$\mathcal{L} = L_0(\boldsymbol{\theta}) + \sum_{\alpha=1}^{N} \left(\lambda_\alpha L_\alpha(\boldsymbol{\theta}) + \tfrac{c}{2}|L_\alpha(\boldsymbol{\theta})|^2\right)$$

Do gradient descent on $\boldsymbol{\theta}$ and a gradient ascent on $\boldsymbol{\lambda}$

For Sparsity: Using just l1 and l2 loss was causing the model to force all weights to zero causing poor accuracy.

Loss function used to facilitate sparsification using the MDMM technique $L_1(\boldsymbol{\theta}) = (target - S(\boldsymbol{\theta}))\sum_i |\theta_i|$



J. Degrave & I. Korchunov

Sparsification and Pruning with MDMM

- The **Baseline model** has **L1/L2 regularizers** to have small weights (basically the penalty method).
- But the penalty method is not the best to do multi-objective optimizations.
- Hence we use **MDMM** to do the multiobjective optimization and use a function to measure the sparsity of the entire model and perform **model sparsification.**

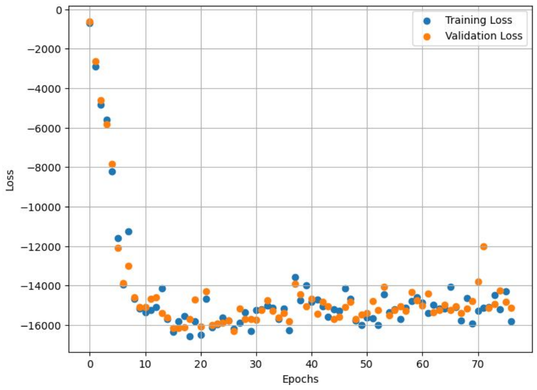
- For multi-objective optimization, we have some **secondary loss functions** along with the **primary loss function** that we are going to minimize.
- Here NLL is the primary loss function and we chose **(target sparsity- current sparsity)*L1** to be the constraint loss function.

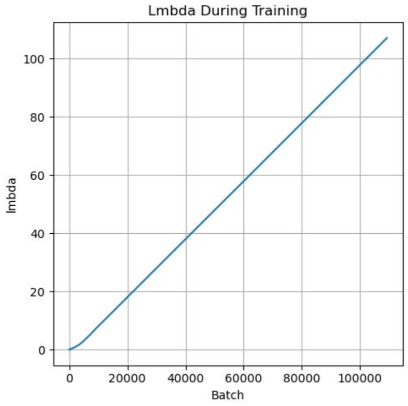
Where **L1 = Sum(|weights|)** and **Sparsity** is the fraction of weights less than some **epsilon** (here we chose epsilon= 1e-3)

66.8% Sparcification

- Final Sparsity of the layers: [0, 0.44, 0.76, 0.9, 0.71]
- Global Sparsity (Mean): 0.668
- epsilon = 1e-3
- So, **66.8 % of the weights** are **below** the **value of 1e-3**
- Final NLL value: -15K

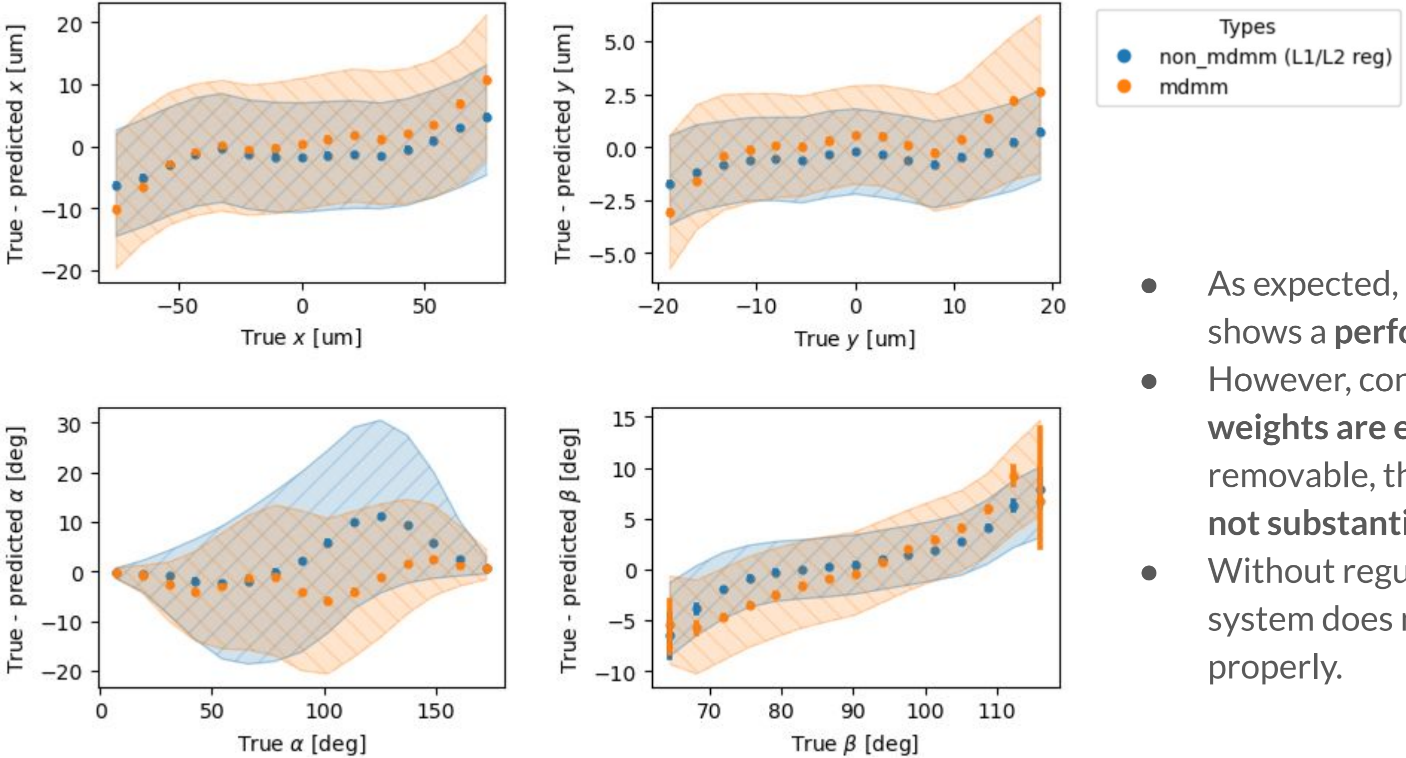
The figure on right shows the (layer-by-layer) evolution of sparsity as training progresses.





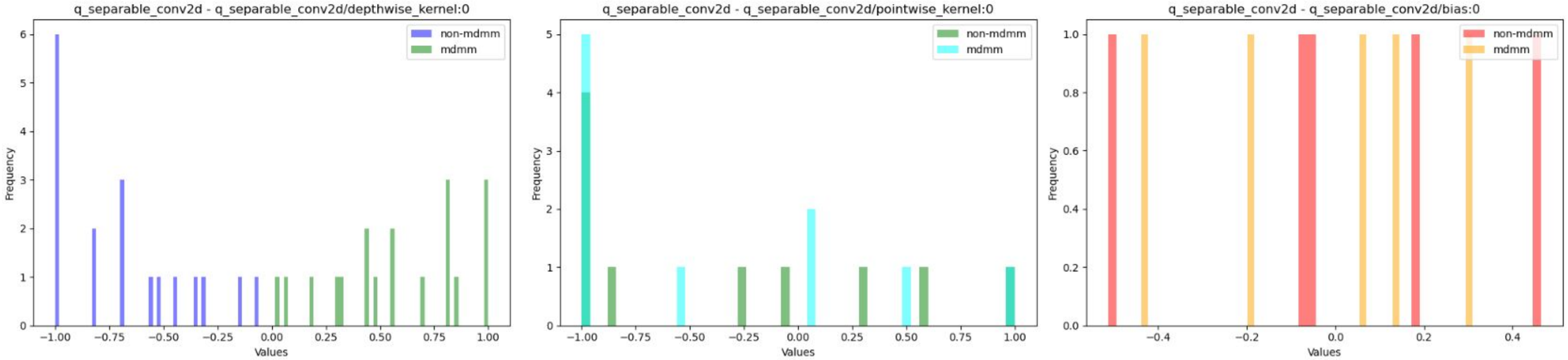
- Majority of sparsification is observed in the **MLP layers.**
- **Convolutional layers** exhibit less sparsification due to reuse of the same filters

Physics performance comparison

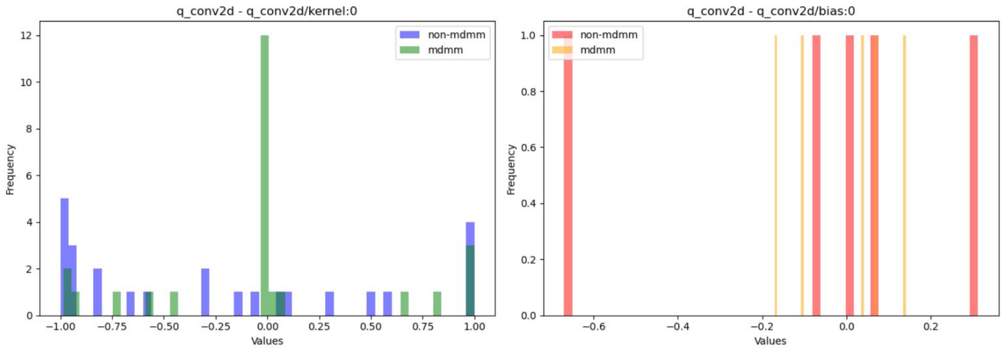


- As expected, the sparse model shows a **performance decrease**.
- However, considering **66.8% of weights are effectively zero** and removable, the **performance drop** is **not substantial**.
- Without regularizers or MDMM the system does not converges properly.

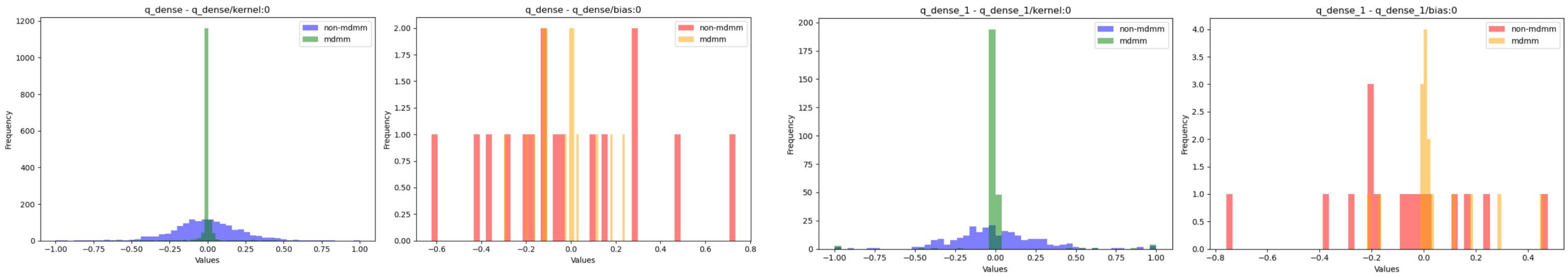
Weights (non-MDMM vs MDMM)



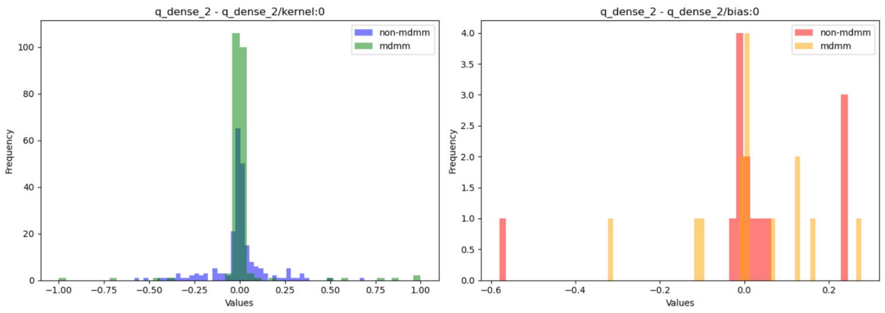
- The **Green** plots shows the model weights **trained with** the **MDMM**.
- We see most of the weights are centered around the zero value
- The **Blue** plots shows the model weights **trained without** the **MDMM**. which is more distributed compared to the green plot.
- The **red** and **orange** plots are **biases** and not our concern at this moment



Weights (non-MDMM vs MDMM)



- Thus there are **better ways** to implement sparsity than **L1/L2 regularizations**.
- We can give a **target sparsity** and achieve that.
- Also we can **target specific layers** and give different target sparsity to each layers
- As, we see the MLP layers high sparsity. So we can try **training with a smaller model** and see how the performances turns out to be



Future Directions

- **Further model quantization** to reduce the number of bits required.
- **Compare the model sparsification** with MDMM and regularizers of different **hyper-parameters.**
- Evaluate various model reduction techniques and its impact on actual **chip performance**, with some loss realistic function taking into account of the chip latencies.
- Apply techniques like **BatchNorm, LayerNorm** etc. to improve model performance.
- We can try **training with a smaller model (**got from **MDMM)** and see how the performances turns out to be.
- Sparsification on **targeted layer by layer** basis with MDMM.

References

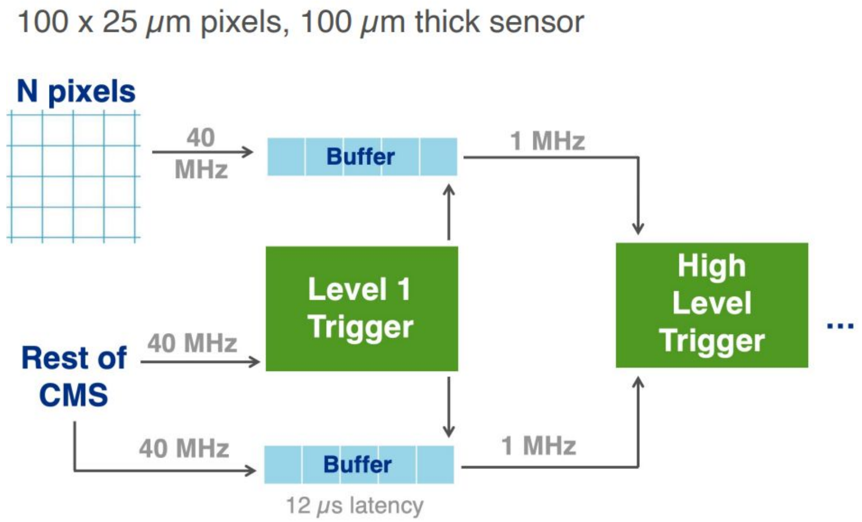
- https://arxiv.org/pdf/2310.02474
- https://arxiv.org/pdf/2312.11676
- https://indico.fnal.gov/event/64625/contributions/295309/attachments/179560/245237/NewPersp-SmartPix_2024.pdf
- https://github.com/jennetd/semiparametric/blob/gauss4d/timeslices-2/neurips-3x3-2conv/draw_from_weights.ipynb
- https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81
- https://indico.slac.stanford.edu/event/7467/contributions/5966/attachments/2869/8024/Dickinson_2023.05.17_LCWS_vf.pdf
- https://www.statlect.com/glossary/log-likelihood

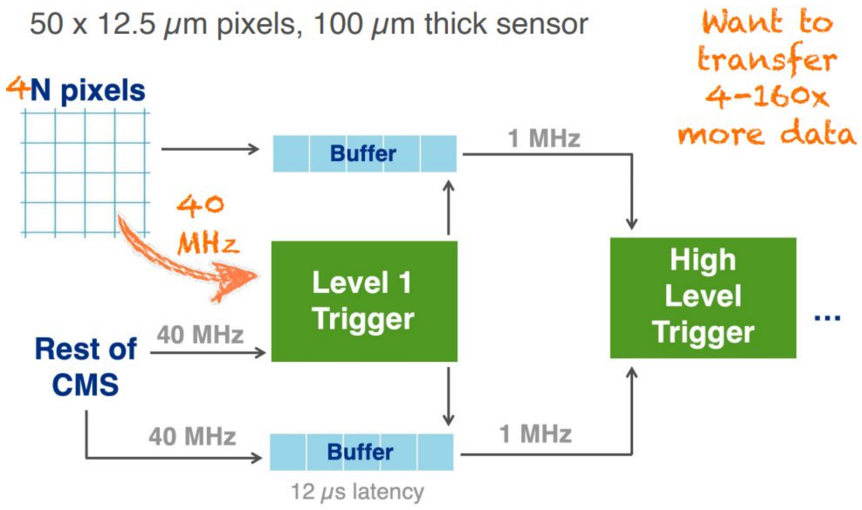
Thank you

Github: https://github.com/ArghyaDas112358/570AI_Final_Project

Backup

What we have? What we will have?





Other Geometries

- # Dataset 2s (Barrel, physical pT)
- dataset2s type1: 50 um x 10 um x 100 um sensor @ 100V
- dataset2s type2: 50 um x 12.5 um x 100 um sensor @ 100V
- dataset2s type3: 50 um x 15 um x 100 um sensor @ 100V
- dataset2s type4: 50 um x 20 um x 100 um sensor @ 100V
- dataset2s type5: 50 um x 25 um x 100 um sensor @ 100V
- dataset2s type6: 100 um x 25 um x 100 um sensor @ 100V
- dataset2s type7: 100 um x 25 um x 150 um sensor @ 175V
-
- # Dataset 3s (Barrel, flat pT)
- dataset3s type1: 50 um x 10 um x 100 um sensor @ 100V
- dataset3s type2: 50 um x 12.5 um x 100 um sensor @ 100V
- dataset3s type3: 50 um x 15 um x 100 um sensor @ 100V
- dataset3s type4: 50 um x 20 um x 100 um sensor @ 100V
- dataset3s type5: 50 um x 25 um x 100 um sensor @ 100V
- dataset3s type6: 100 um x 25 um x 100 um sensor @ 100V
- dataset3s type7: 100 um x 25 um x 150 um sensor @ 175V
-
- # Dataset 4s (End-caps, physical pT)
- dataset4s type1: 50 um x 10 um x 100 um sensor @ 100V
- dataset4s type2: 50 um x 12.5 um x 100 um sensor @ 100V
- dataset4s type3: 50 um x 15 um x 100 um sensor @ 100V
- dataset4s type4: 50 um x 20 um x 100 um sensor @ 100V
- dataset4s type5: 50 um x 25 um x 100 um sensor @ 100V
- dataset4s type6: 100 um x 25 um x 100 um sensor @ 100V
- dataset4s type7: 100 um x 25 um x 150 um sensor @ 175V

Variables (code)

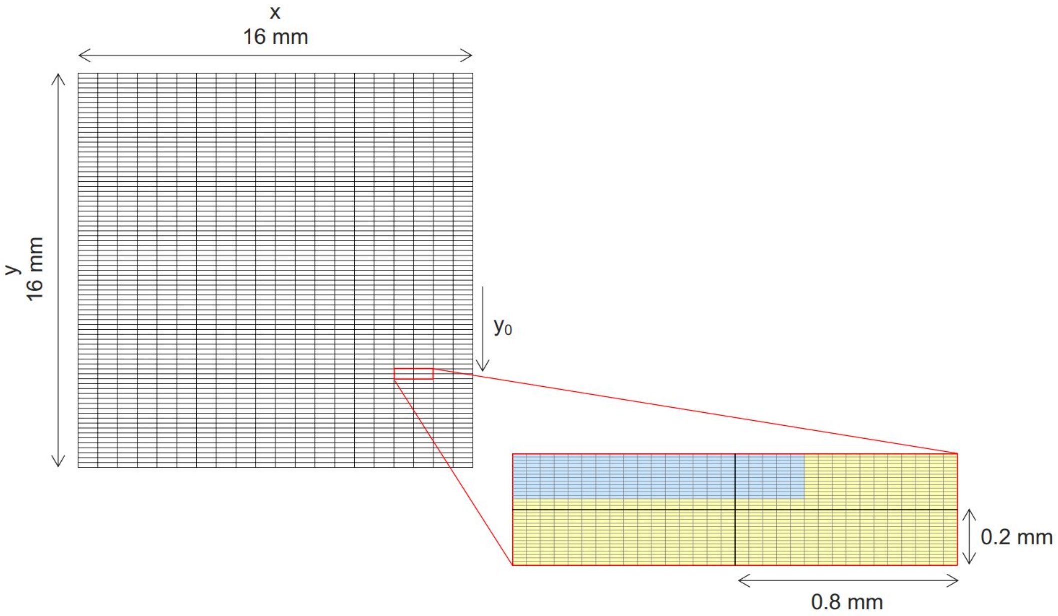
```
df['sigmax'] = abs(df['M11'])
df['sigmay'] = np.sqrt(df['M21']**2 + df['M22']**2)
df['sigmacotA'] = np.sqrt(df['M31']**2+df['M32']**2+df['M33']**2)
df['sigmacotB'] = np.sqrt(df['M41']**2+df['M42']**2+df['M43']**2+df['M44']**2)
'''df['cov'] = np.sqrt(df['M11']*df['M21'])'''

df['pullx'] = (df['xtrue']-df['x'])/df['sigmax']
df['pully'] = (df['ytrue']-df['y'])/df['sigmay']
df['pullcotA'] = (df['cotAtrue']-df['cotA'])/df['sigmacotA']
df['pullcotB'] = (df['cotBtrue']-df['cotB'])/df['sigmacotB']
```

```
fig, axes = plt.subplots(2,2,sharex=True,sharey=True,figsize=(8,6))
pull_plot(axes[0][0],'pullx',r'$x$ pull')
pull_plot(axes[0][1],'pully',r'$y$ pull')
pull_plot(axes[1][0],'pullcotA',r'$\cot\alpha$ pull')
pull_plot(axes[1][1],'pullcotB',r'$\cot\beta$ pull')
```

```
Mean -0.029727496110678025
Sigma -0.9297218351149932
Mean -0.10882627175349704
Sigma 0.8644523573719014
Mean -0.12909672012398107
Sigma -0.8106461105972753
Mean -0.2167140557412369
Sigma 0.8842340287636578
```

Pixel AV: https://cds.cern.ch/record/687440?ln=en

- Provides an accurate model of charge deposition, particularly from hadronic tracks.
- Includes a realistic mapping of the electric field.
- Incorporates an established model for charge drift physics.
- Accounts for electronic noise, response, and threshold effects.
- Models the time evolution of drift and induced currents within the pixel sensor.



x
16 mm
y
16 mm
y0
0.2 mm
0.8 mm

Negative Log-Likelihood

$$f(x_i) = \left(2\pi\sigma^2\right)^{-1/2} \exp\left(-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right)$$

$$f(x_i;\mu,\sigma^2) = \left(2\pi\sigma^2\right)^{-1/2} \exp\left(-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right)$$

$$\begin{aligned} L(\theta;\xi) = f(\xi;\theta) &= \prod_{i=1}^{n} f_X(x_i;\mu,\sigma^2) \\ &= \prod_{i=1}^{n}\left(2\pi\sigma^2\right)^{-1/2} \exp\left(-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right) \\ &= \left(2\pi\sigma^2\right)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right) \end{aligned}$$

$$\begin{aligned} l(\theta;\xi) &= \ln[L(\theta;\xi)] \\ &= \ln\left[\left(2\pi\sigma^2\right)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right)\right] \\ &= \ln\left[\left(2\pi\sigma^2\right)^{-n/2}\right] + \ln\left[\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right)\right] \\ &= -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \\ &= -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\left(\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \end{aligned}$$