Machine Learning for Sextupole Surrogate Models of the CLIC Final-Focus System

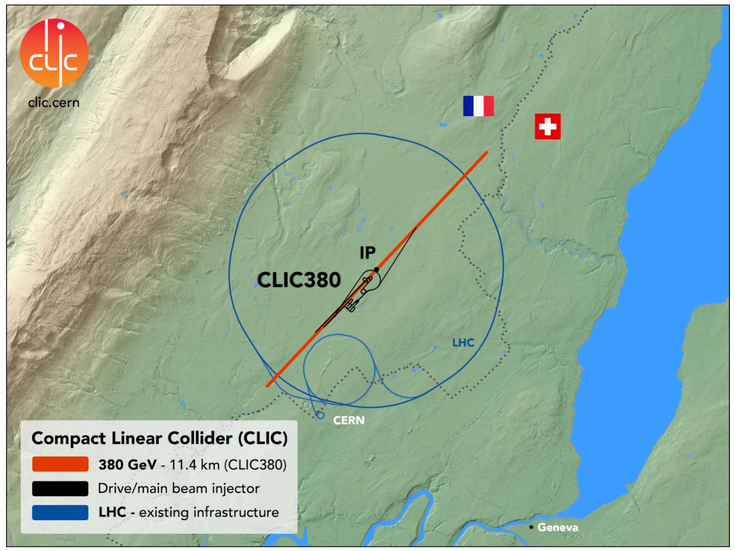
Jim Ögren

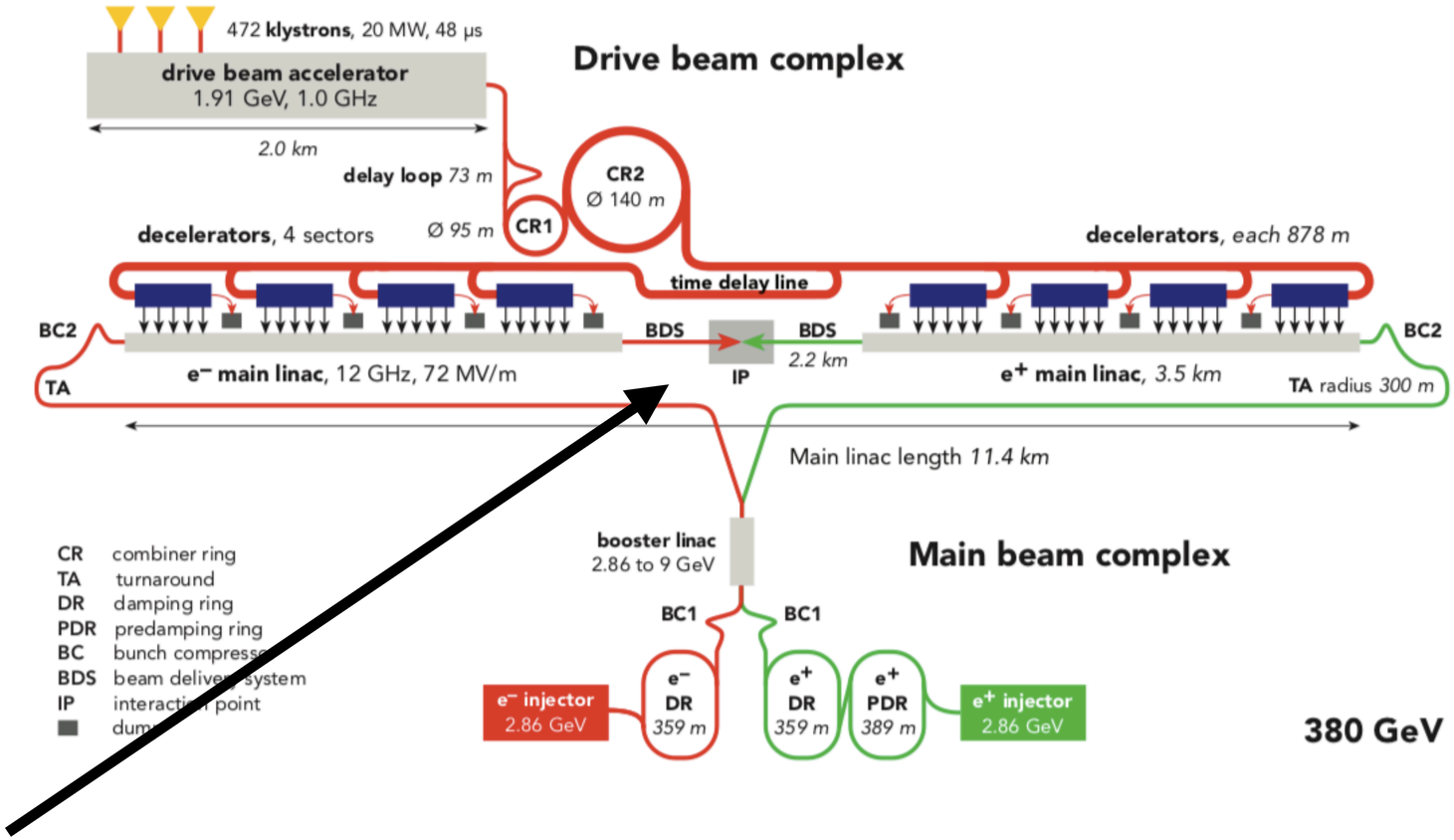
International Workshop on Future Linear Colliders (LCWS19)
October 28—November 1, 2019
Sendai, Japan

Acknowledgements: Chetan Gohil, Daniel Schulte

Compact Linear Collider (CLIC)

Footprint





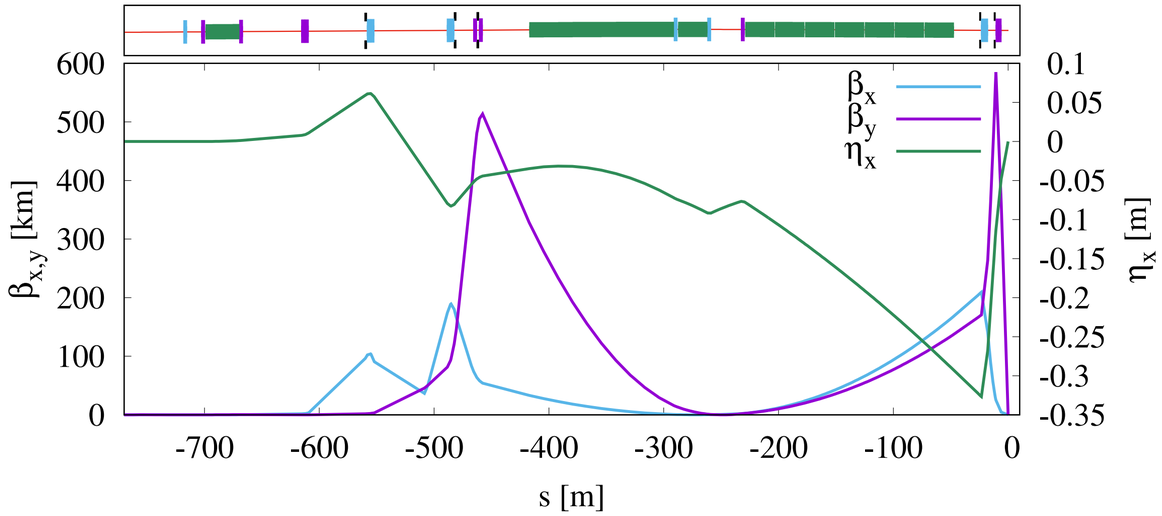
The beam delivery system - Last 2.2 km of the main linacs

- Collimation section
- Final-focus system (last 780 m)

The 380 GeV CLIC FFS

The final-focus system (FFS):

- Local chromaticity scheme
- 780 m total length
- 20 quadrupole magnets
- 22 Beam-position monitors (BPMs)
- 6 sextupole magnets
- 2 octupole magnets
- 385 m bending magnets



Norm. emittance (end of linac) $\gamma\epsilon_x/\gamma\epsilon_y$	[nm]	900 / 20
Norm. emittance (IP) $\gamma\epsilon_x/\gamma\epsilon_y$	[nm]	950 / 30
Beta function (IP) β^*_x/β^*_y	[mm]	8.2 / 0.1
Target IP beam size σ^*_x/σ^*_y	[nm]	149 / 2.9
Bunch length σ_z	[μm]	70
rms energy spread δ_p	[%]	0.35
Bunch population N_e	[10^9]	5.2
Number of bunches n_b		352
Repetition rate f_rep	[Hz]	50
Luminosity $\mathcal{L}_\mathrm{total}$	[$10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$]	1.5
Peak luminosity $\mathcal{L}_{1\%}$	[$10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$]	0.9

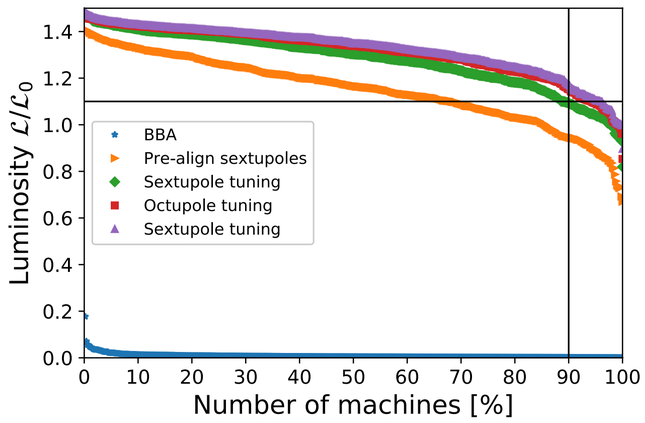
Luminosity calculation:

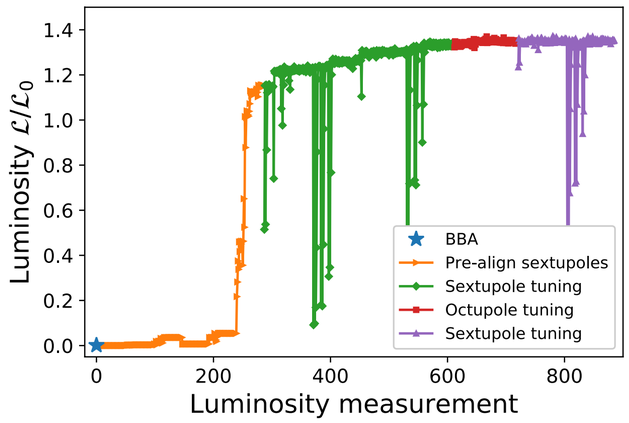
N = particles per bunch
n_b = number of bunches
f_r = repetition rate
σ = beam size at collision point
H = correction factor (hour glass effect, disruption)

$$L = H\frac{N^2 n_b f_r}{\sigma_x \sigma_y}$$

Tuning simulations, static imperfections

- Single-beam: only half of FFS, beam mirrored at IP
- Transverse misalignments, rolls and magnetic strength errors
- Monte Carlo simulations. Goal: 90% of machines successfully tuned
- Tuning time is essential for collider performance





J. Ogren, A. Latina, R. Tomas and D. Schulte, *Tuning of the CLIC 380 GeV Final-Focus System with Static Imperfections*, CERN-ACC-2018-0055, CLIC-Note-1141.

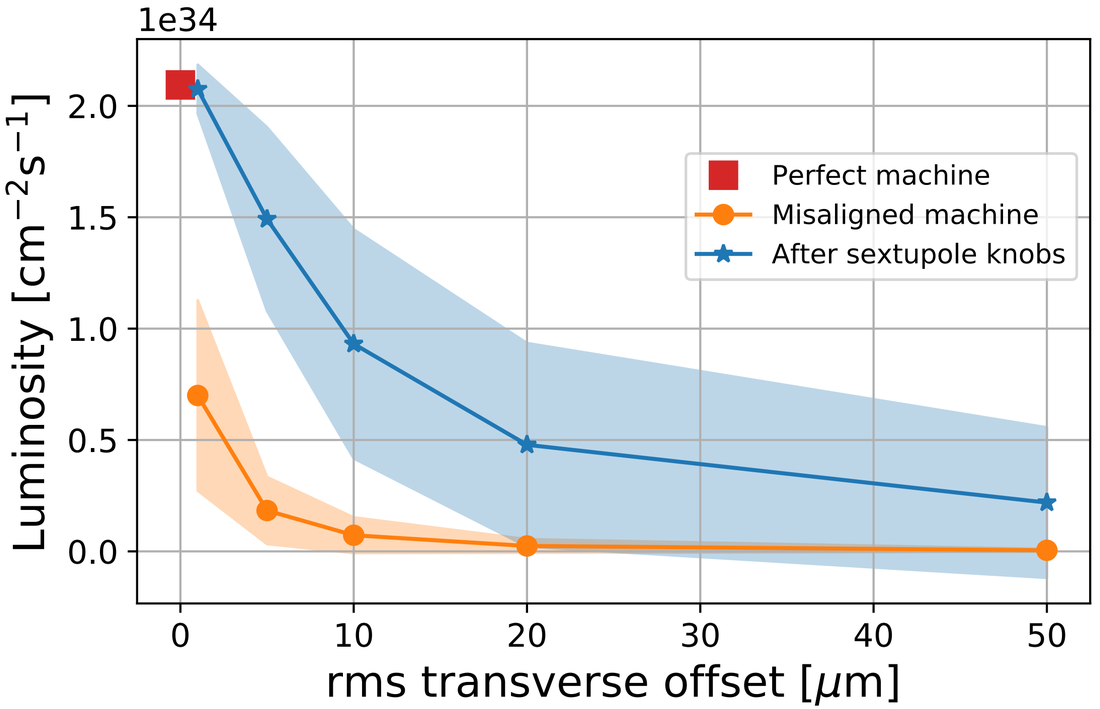
Static imperfections	Specified tolerance (rms error)	Elements
Resolution	20 nm	BPMs
Transverse misalignments	10 μm	BPMs, quadrupoles, multipoles
Roll errors	100 μrad	BPMs, quadrupoles, multipoles
Relative strength error	10^{-4}	Quadrupoles, multipoles

Machine learning application?

First objective: create a surrogate model to replace computationally intensive simulations

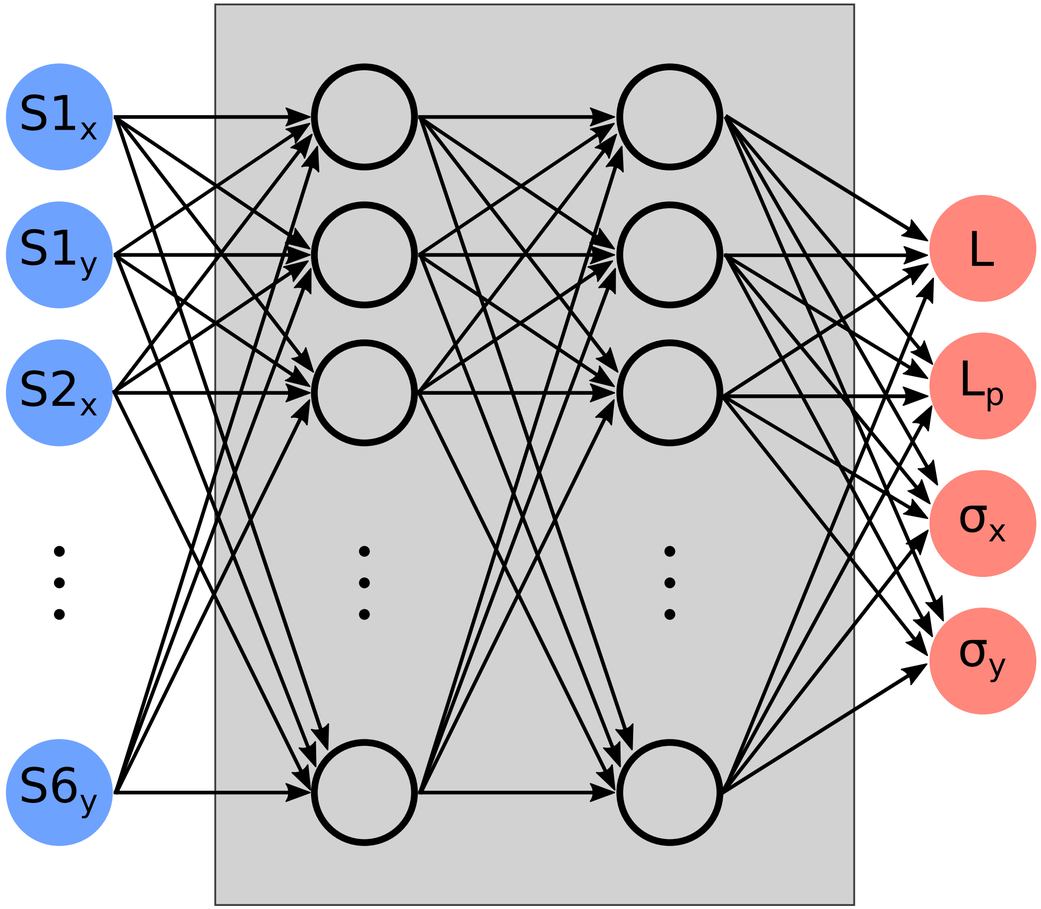
Single imperfection: sextupole transverse offsets

- Sextupole transverse offsets
  - Big impact on luminosity
- Different approaches
  - Orthogonal knobs
  - Random walk
- Efficient tuning is crucial
- Simulations are time consuming
- Make a surrogate model
  - Analytical model?
  - Nonlinear system
  - Synchrotron radiation
  - Beam-beam effect with large disruption
  - Make use of machine learning?
  - Artificial neural networks?



Sextupole surrogate model

- Model sextupole transverse positions to Luminosity. Goal: fast estimator.
- Supervised, deep learning
- Artificial Neural Networks



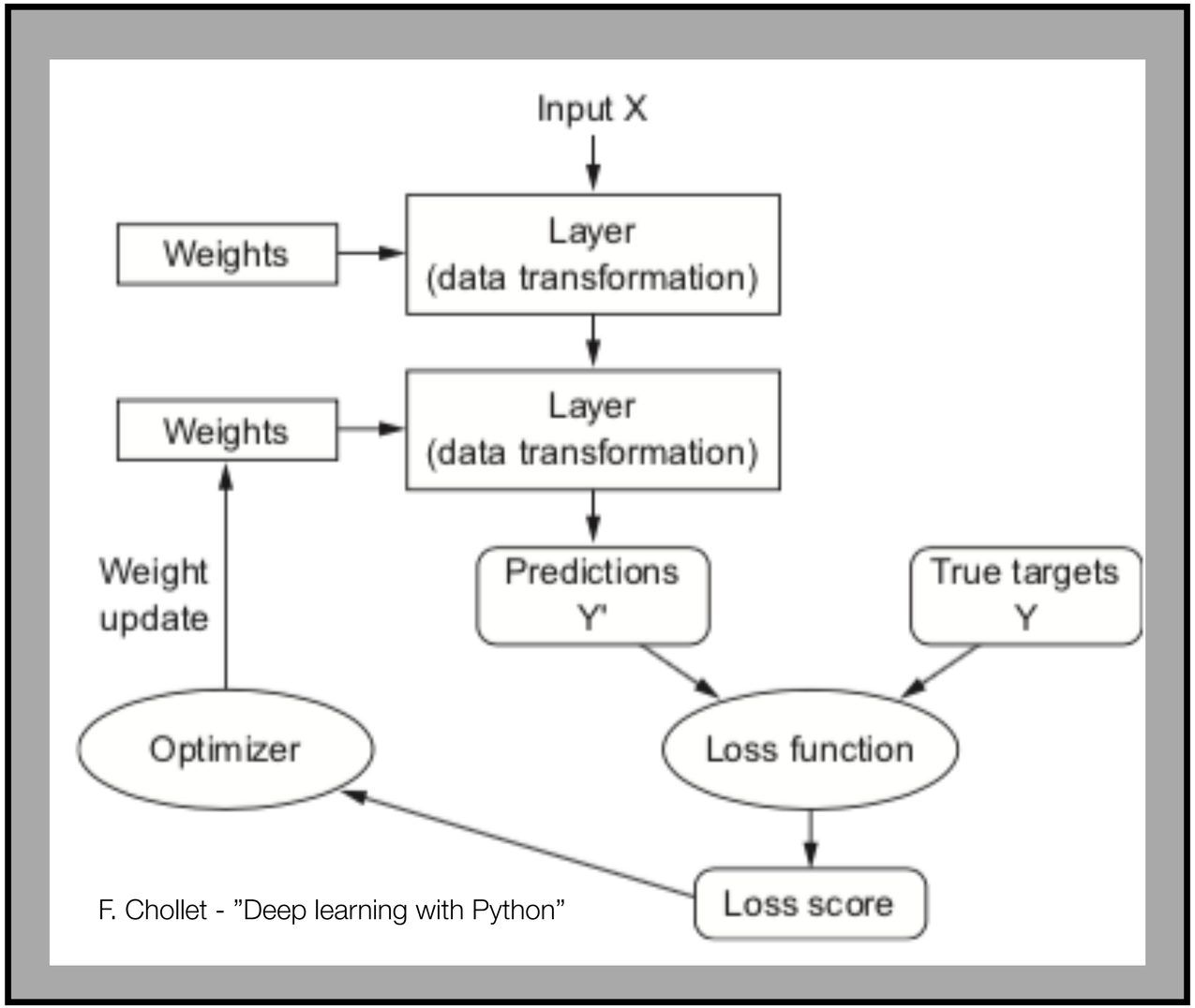
Data generation

- Tracking in PLACET, beam-beam in GUINEA-PIG
- Simulate perfect machine with sextupole transverse offsets (5, 10, 20 μm rms)
- 1 run = 10 random cases (less then 20 mins) 10,000 jobs at the time
- Generated about 450,000 data points

Machine Learning:

- Deep learning with artificial neural networks
- TensorFlow and Python library Keras

Model training



- Adjust weights and biases for each node to minimize the loss function
- Loss function: e.g. mean square error
- Algorithm: *backpropagation of errors*
  (gradient descent) adjust weights to minimize loss
- Implemented in TensorFlow
- Python Keras library to interface

- Split data: training (80%), testing (20%)
- Testing data only used for model evaluation

Model training

Simple example:

```
import tensorflow as tf
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(12, activation='relu', input_shape=(x.shape[1],)))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(4))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

history=model.fit(x, y, epochs=400, batch_size=50, validation_split=0.1)
```

- Sequential model = feedforward network
- Dense layer = all nodes are connected to all nodes in the next layer
- Specify number of nodes in each layer and activation function
- Batch_size = number of data points in each iteration of backprogation
- 1 epoch = 1 loop over the full data set

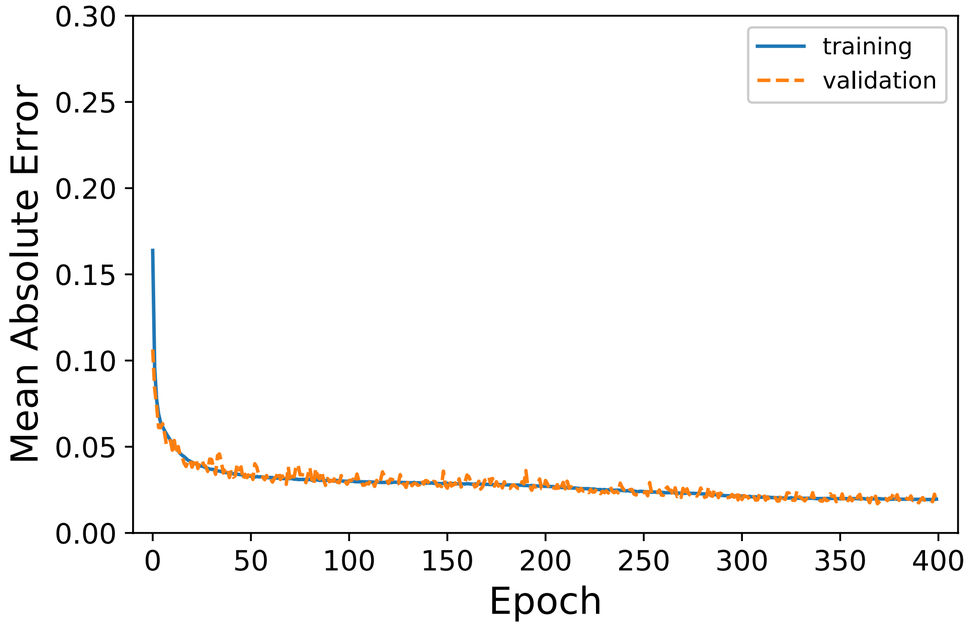
Model training

Simple example:

```
import tensorflow as tf
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(12, activation='relu', input_shape=(x.shape[1],)))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(12, activation='relu'))
model.add(tf.keras.layers.Dense(4))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

history=model.fit(x, y, epochs=400, batch_size=50, validation_split=0.1)
```

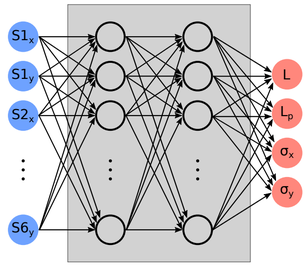
Validation split to monitor overfitting

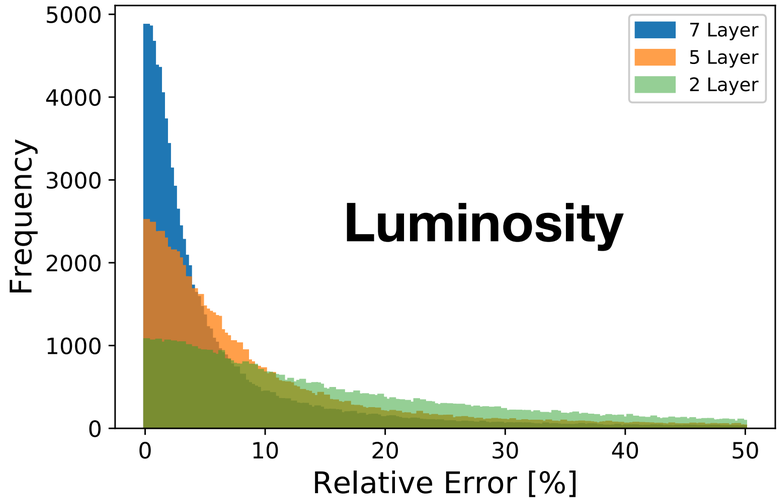


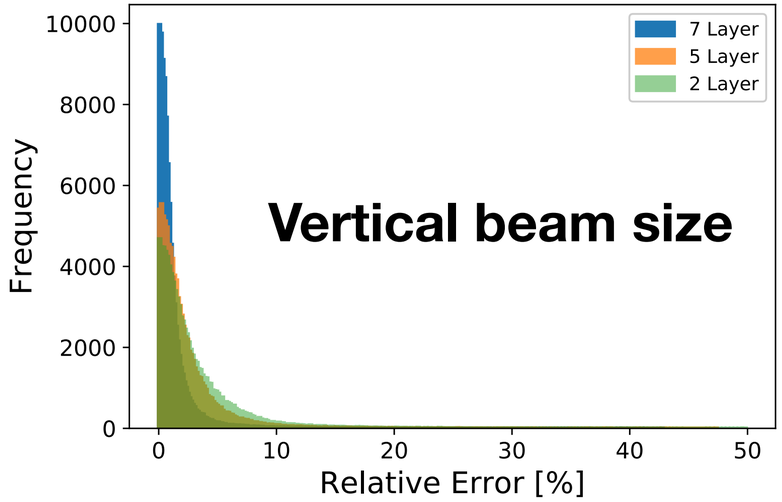
- Sequential model = feedforward network
- Dense layer = all nodes are connected to all nodes in the next layer
- Specify number of nodes in each layer and activation function
- Batch_size = number of data points in each iteration of backprogation
- 1 epoch = 1 loop over the full data set

Model performance

	Luminosity			Vertical beam size		
	2 Layers	5 Layers	7 Layers	2 Layers	5 Layers	7 Layers
Mean(|Rel_error|) [%]	29.1	11.7	6.5	3.0	2.3	1.2
Std(|Rel_error|) [%]	62.4	20.1	11.1	3.5	2.8	2.2
90% less than [%]	64.7	27.4	15.8	6.7	5.1	2.6



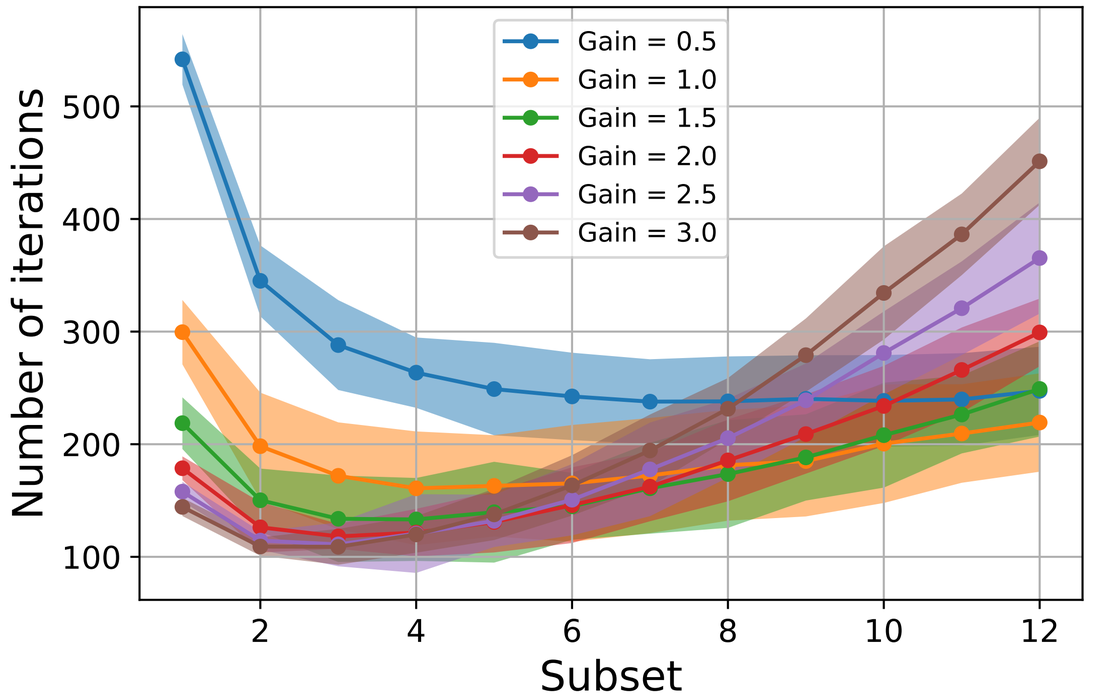




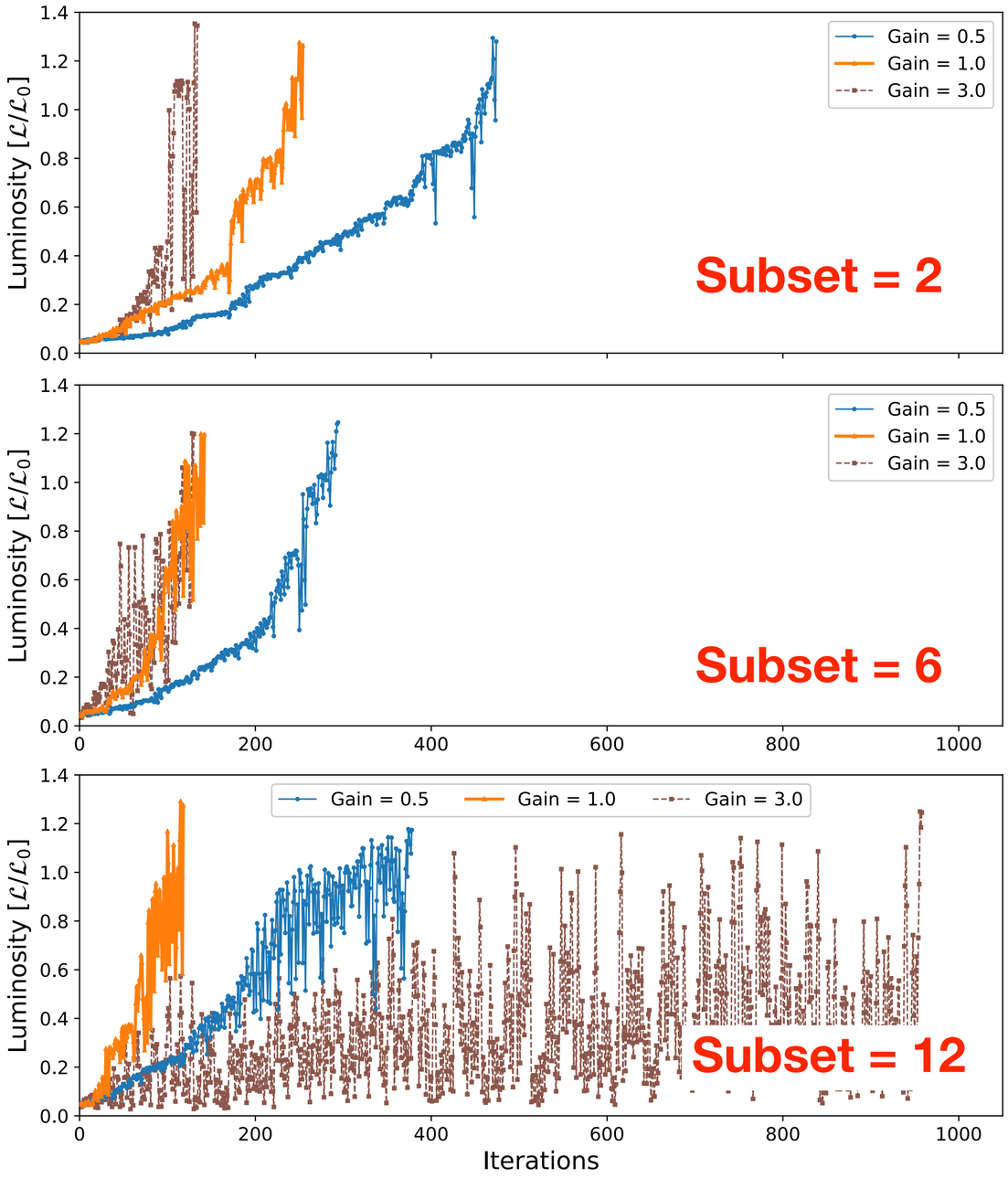
- It seems difficult to get the mean error below 5-6%
- A model in a more narrow range performs better
- Part of it comes from Luminosity uncertainty (~1%)
- Accuracy is not the most crucial aspect
- A model that correctly characterizes the behavior is very useful

Random walk algorithm

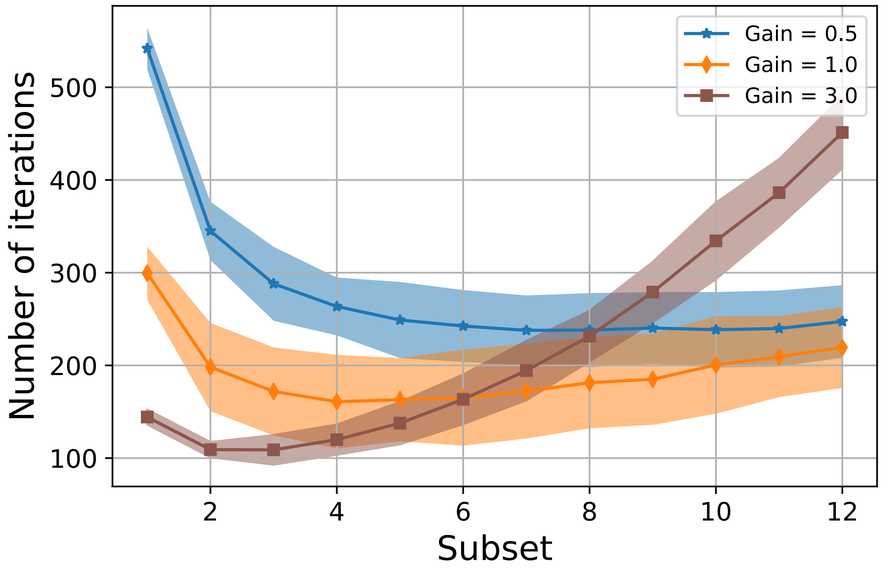
- ML model: 1000 iteration random walk tuning takes a few seconds
- Full-scale simulation: 1000 iterations random walk tuning takes ~8h
- Use ML model to optimize algorithm
- For each setting: 100 different random seeds, each tuned 100 times
- Random walk:
  - Select a subset out of the 12 DOF
  - Make steps in random direction:
    gain[-1, -0.5, 0.5, 1]*



Full simulation



- Test random walk parameters on full simulation
- A few example

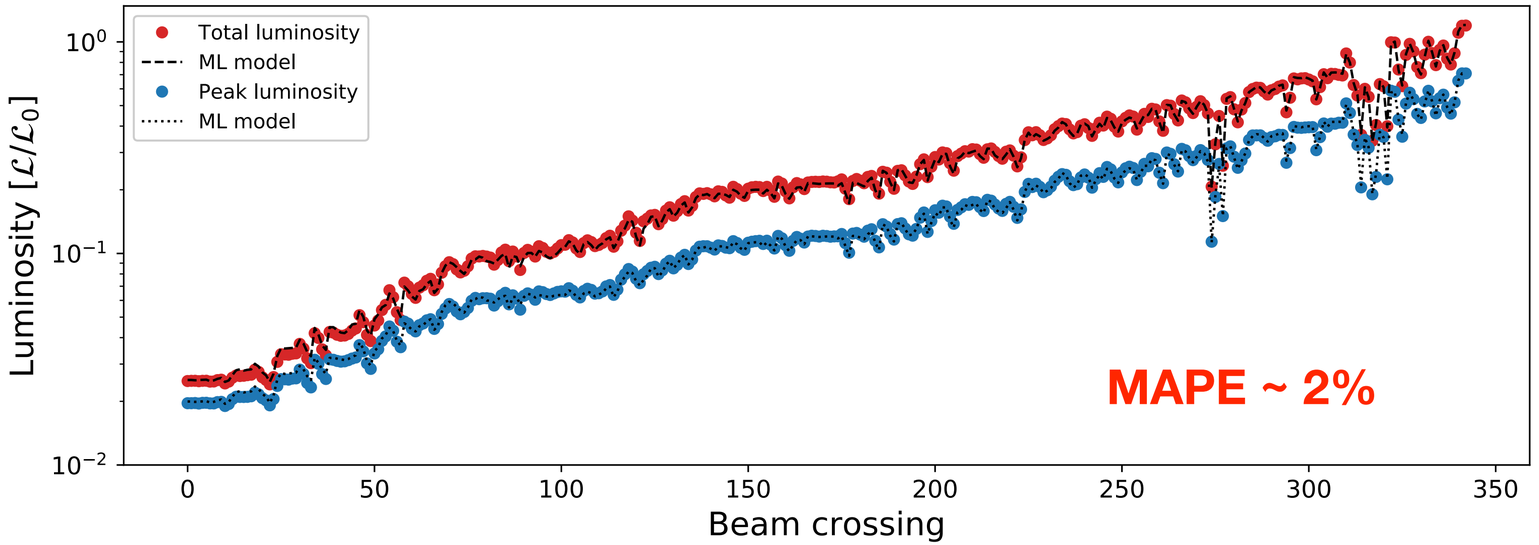


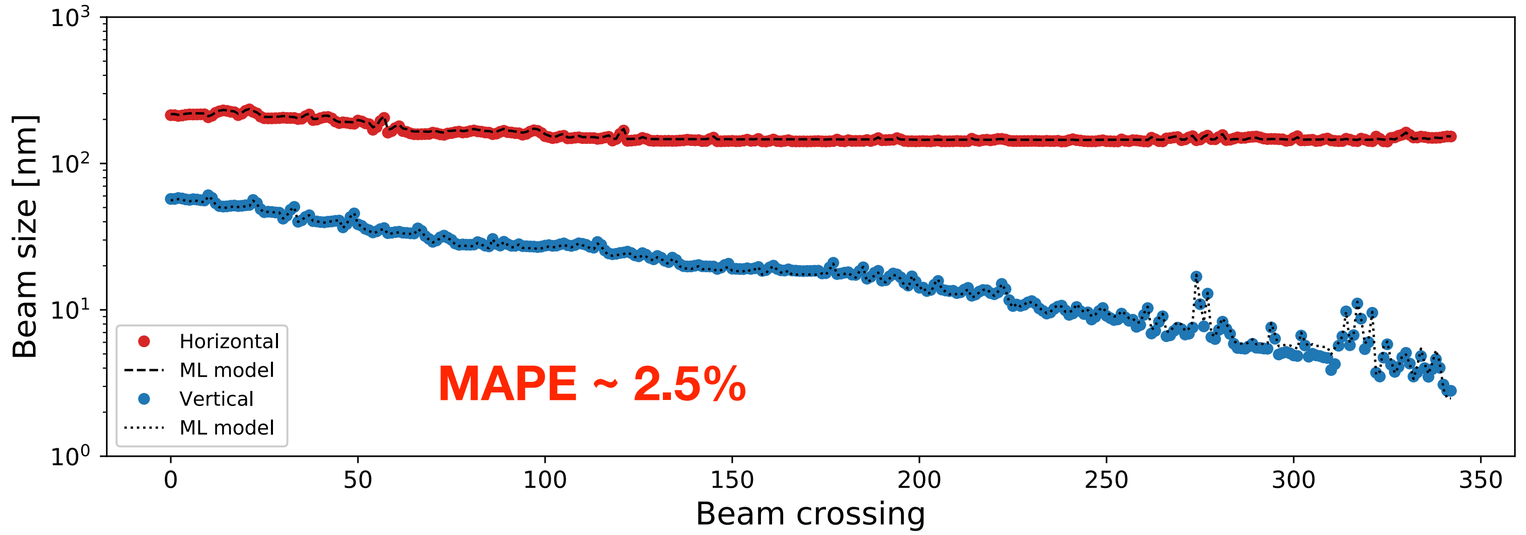
Compare with full simulation

Simulation

- Tuning of the perfect machine with sextupole transverse offsets only
- Use the normal tuning knobs and full-scale tracking (100,000 macroparticles)
- At each step: evaluate luminosity from ML model as well and save to file
- To evaluate predictive performance, mean absolute percentage error (MAPE):

$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i-\tilde{y}_i}{y_i}\right|$$

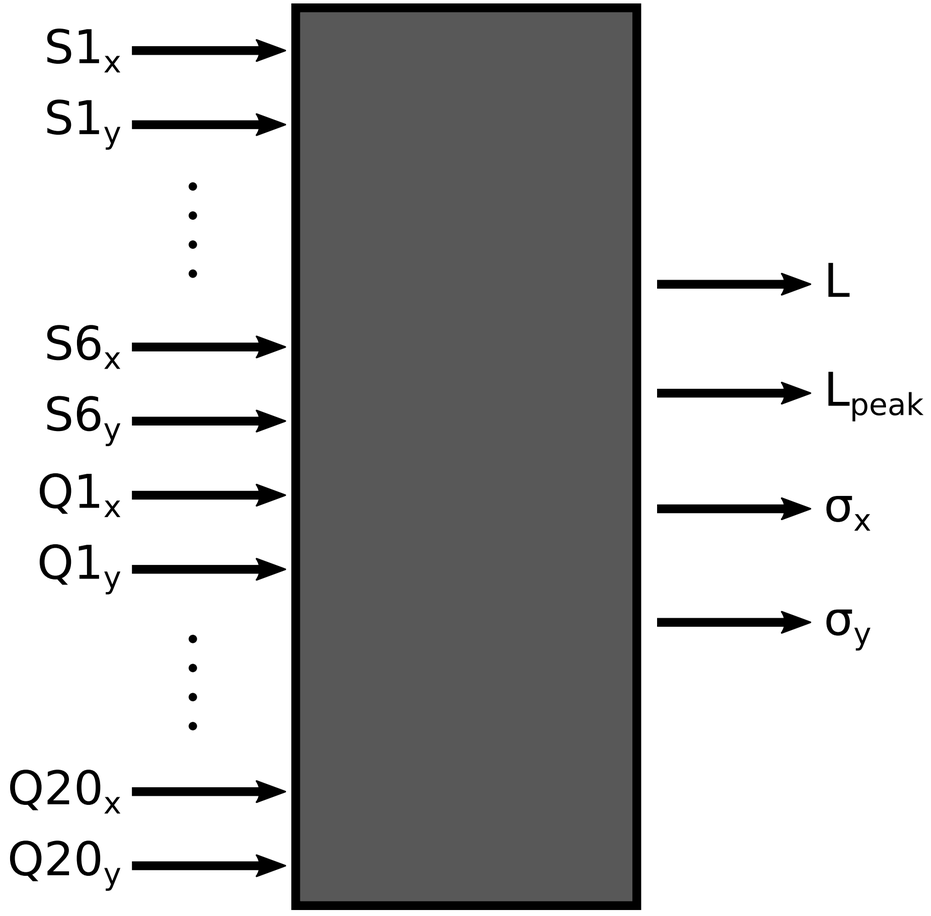




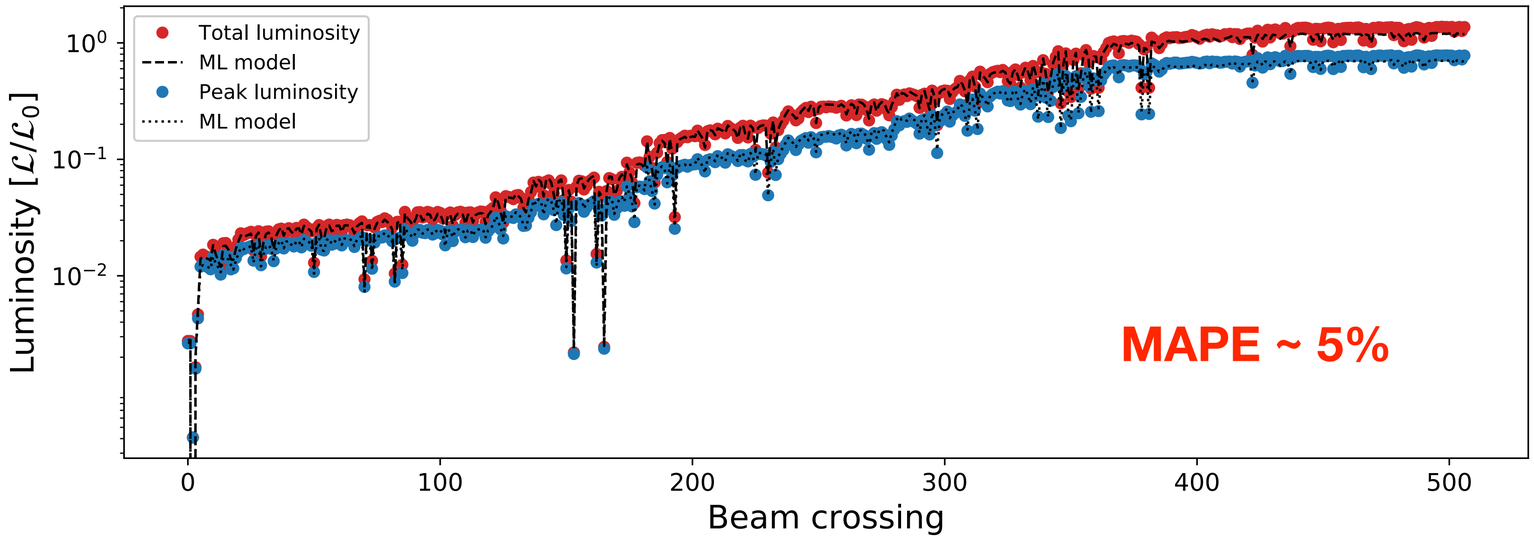
Quadrupoles and sextupole model

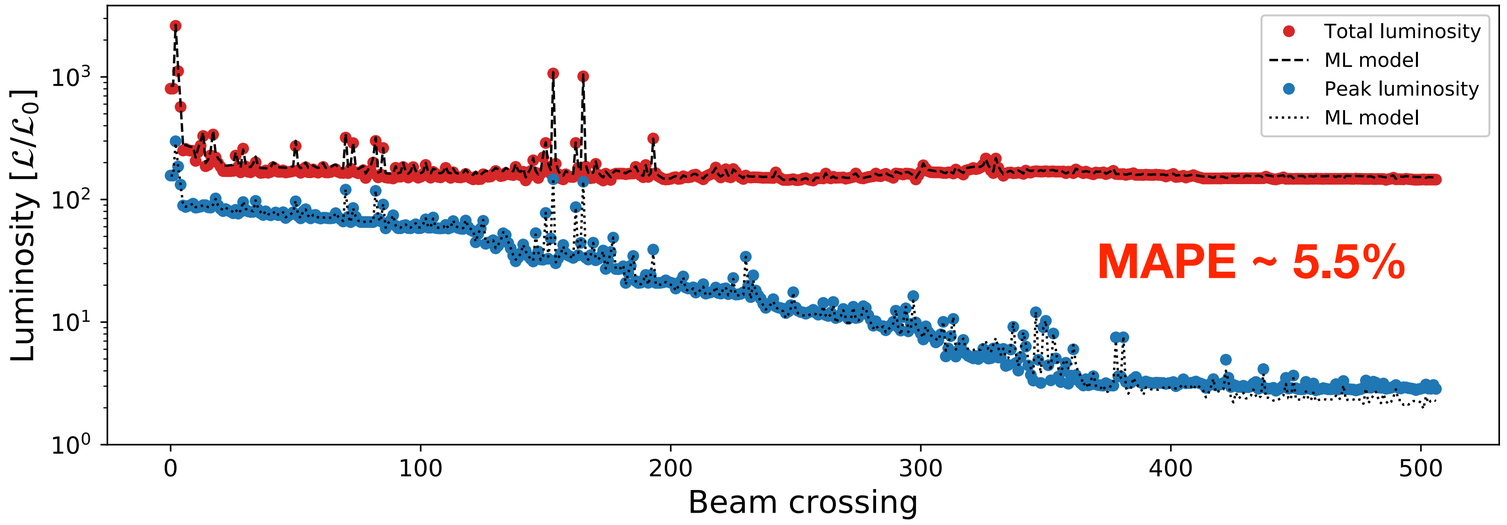
Extended model

- Sextupole transverse position and quadrupole transverse position
- 20 Quadrupoles, 12 sexupoles => 52 input parameters
- Train networks that map offsets to L, LP, σx, σy
- Same procedure as before
- Optimized random walk parameters on ML model



Compare with full simulation



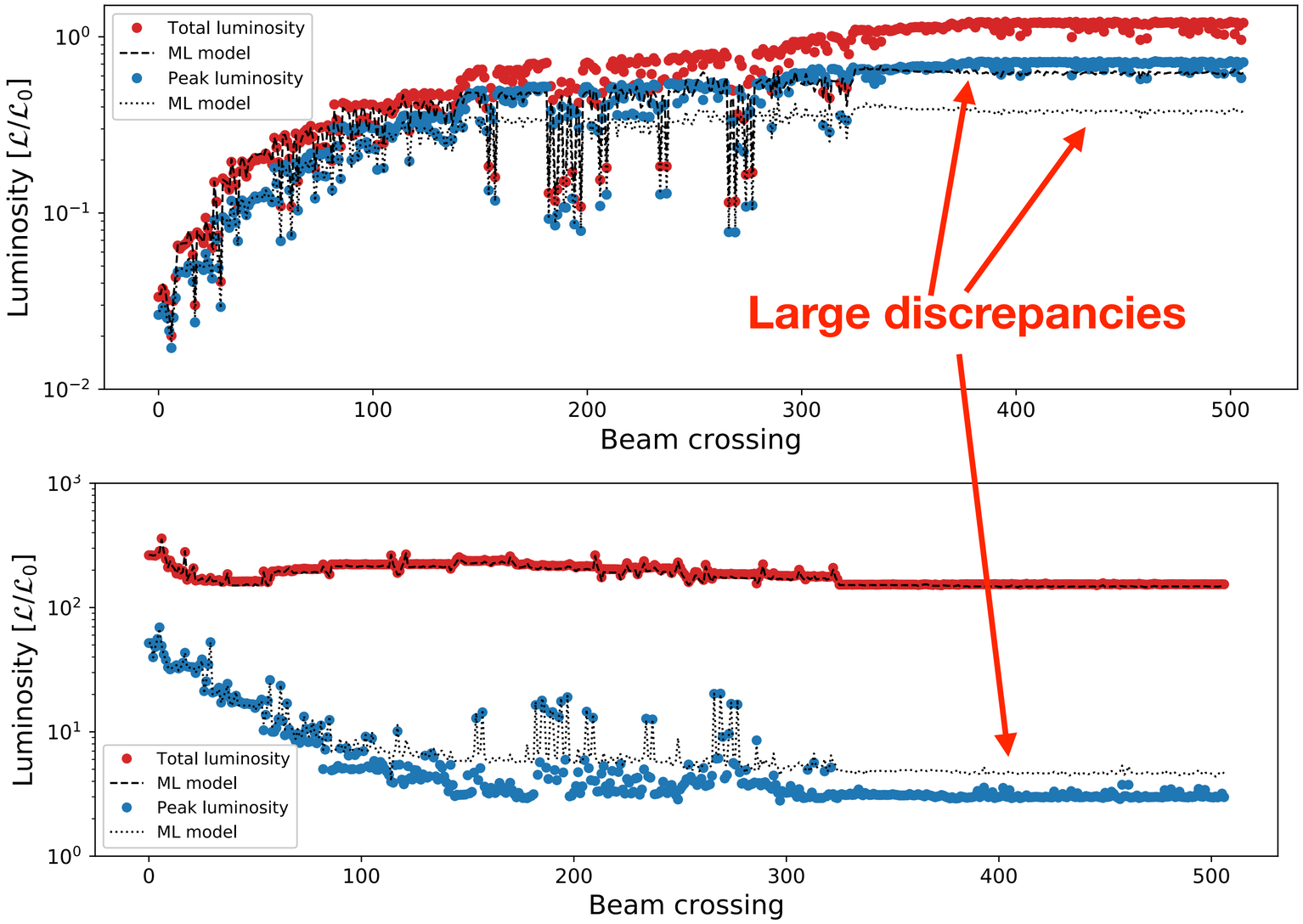


Random walk simulation

- Moving quadrupoles and sextupoles
- At each step: evaluate luminosity from ML model as well
- ML model manages to capture the characteristics pretty well

Not always successful

- In some cases: large discrepancy
- Model range limited
- Improvement on model
- Use more data



Conclusions

- Deep learning and artificial neural networks
  - Trained sextupole surrogate model on a large set of simulation data
  - Accurate in quite a large range
  - Model was cross-checked with full-scale simulations

- Tuning random walk hyperparameters
  - Model is very fast to evaluate
  - Scan over hyperparameters with a large set of examples
  - Model was useful and results were used in the full tuning simulation

- Extended model to include quadrupole offsets
  - Model worked quite well in a limited range but gave large discrepancies in some cases
  - Will try to improve model by training on more data

- Future work
  - Include more imperfections
  - Testing other Machine Learning techniques
  - E.g. reinforcement learning