Estimating Hidden Fluctuating Magnetic Field Gradient with Spin Qubits

Presentation made by Athene Demuth, Arnulf, Oliver Liebe, and Rune Kutchinsky

14. June, 2023.

UNIVERSITY OF COPENHAGEN

Motivation

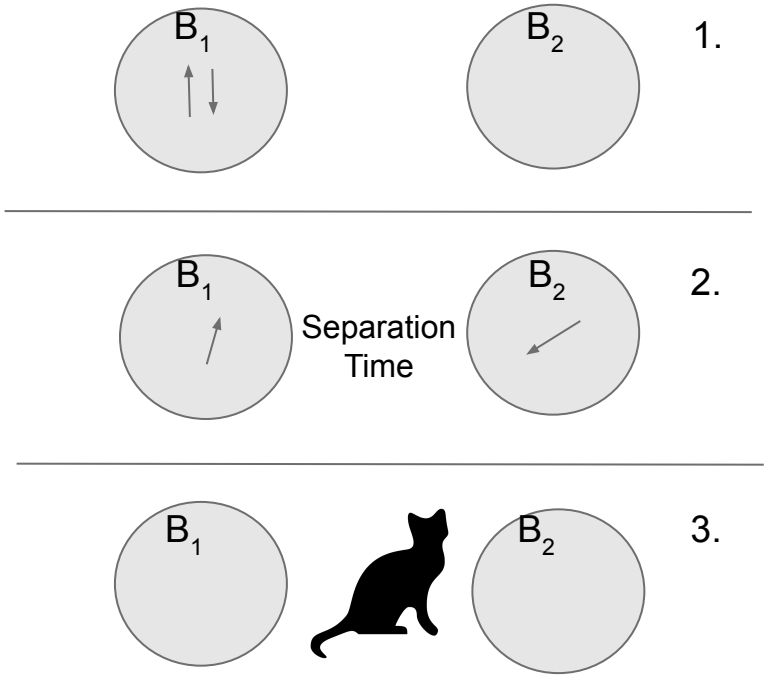
Estimating dBz (Overhauser Gradient):

- Bayes too slow
  - Due to method + too many measurements
- Attempt a Neural Network (NN)
  - Works on OPX
- Downsampling
  - Preselection of separation times
  - (towards live time selection)
- More ML methods
  - Convolutional NN
  - Boosted Decision Tree
  - (Generative Adversarial Network)

Nothing has been implemented on the OPX

The Data: Quantum Explanation, (Heavily Simplified)

1. Two electrons in a dot
   - => Only opposite spins
2. To two dots
   - Separation Time
   - B_z difference (random)
   - Spins rotate independently
3. Recombination
   - Success if opposite spins
   - Bool
4. Repeat 100 times

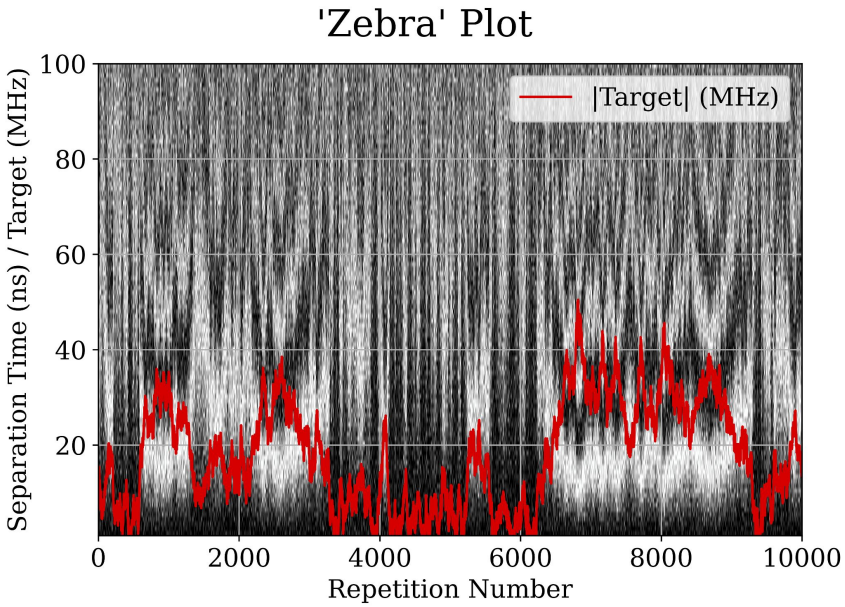


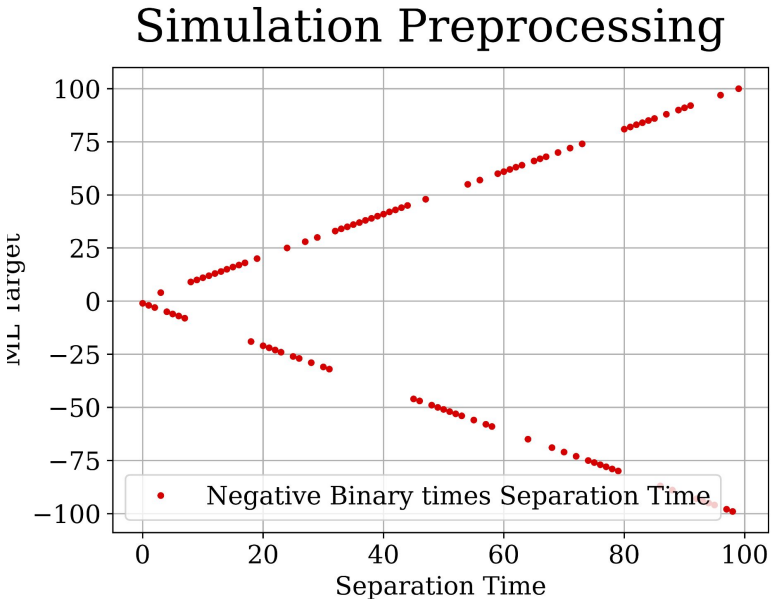
GOAL: Estimate Magnetic Field Difference

The Data: Simulation and Preprocessing

Zebra plots

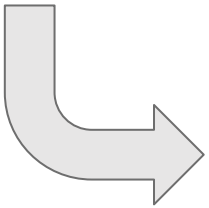
Linear transformation





Constraints and the project plan

- To work on an FPGA system (outside project scope)
- Smallest possible implementation
- Specific activation functions (Relu, Elu)
- No FFT/Convolutions

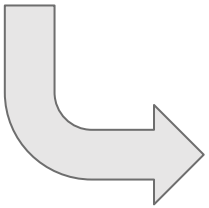


Three-step plan:

- 1. Regression to dBz
- 2. Reduction of data
- 3. Reinforcement

Constraints and the project plan

- To work on an FPGA system (outside project scope)
- Smallest possible implementation
- Specific activation functions (Relu, Elu)
- No FFT/Convolutions



Three-step plan:

- 1. Regression to dBz ✅
- 2. Reduction of data ✅
- 3. Reinforcement ~

Negative Log Likelihood Loss Function

Producing a Generative Adversarial Network, to predict the next most important separation time.

The Negative Log Likelihood also produces the uncertainty.

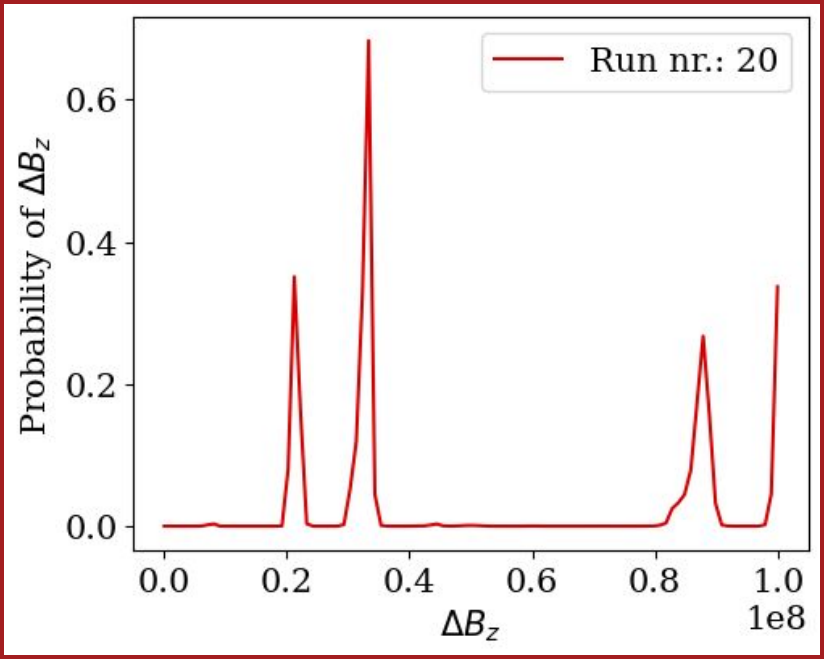
Online estimation of the next separation time

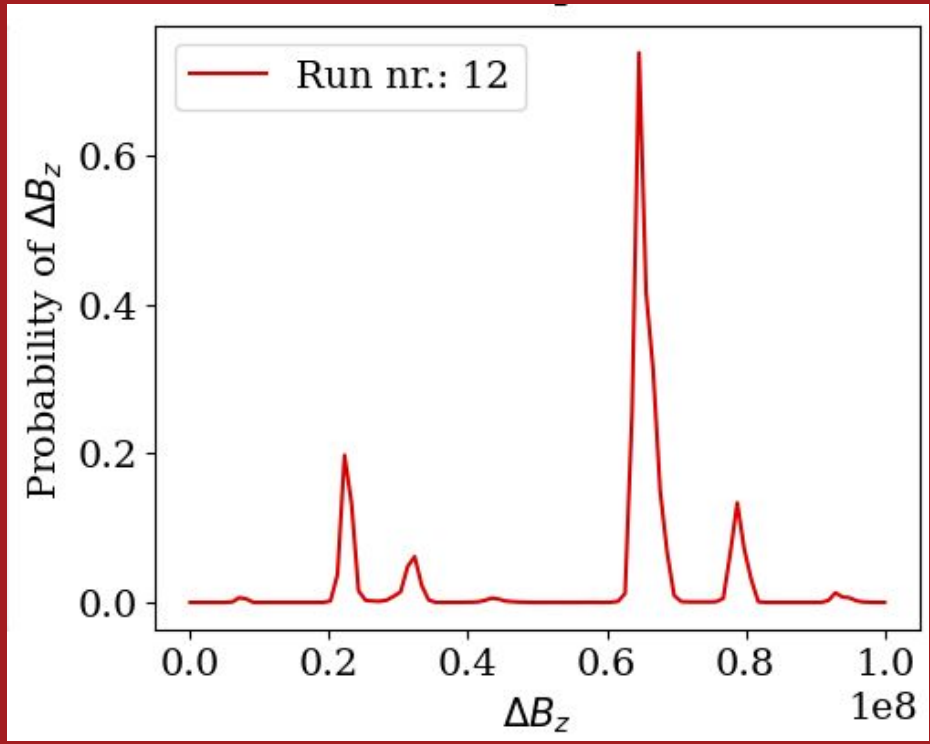
$$NLLH = -\frac{(y_{true} - \mu)^2}{2\sigma^2} + \frac{1}{2}\log\left(\frac{1}{\sigma^2}\right)$$

Benchmark - What has been done so far?

Our own implementation of the slower Bayesian probability distribution calculations.

How does it work?





What have we done?

Custom Negative Log Likelihood Function

Feed Forward Neural Network
- Downsampling

Convolutional Neural Network

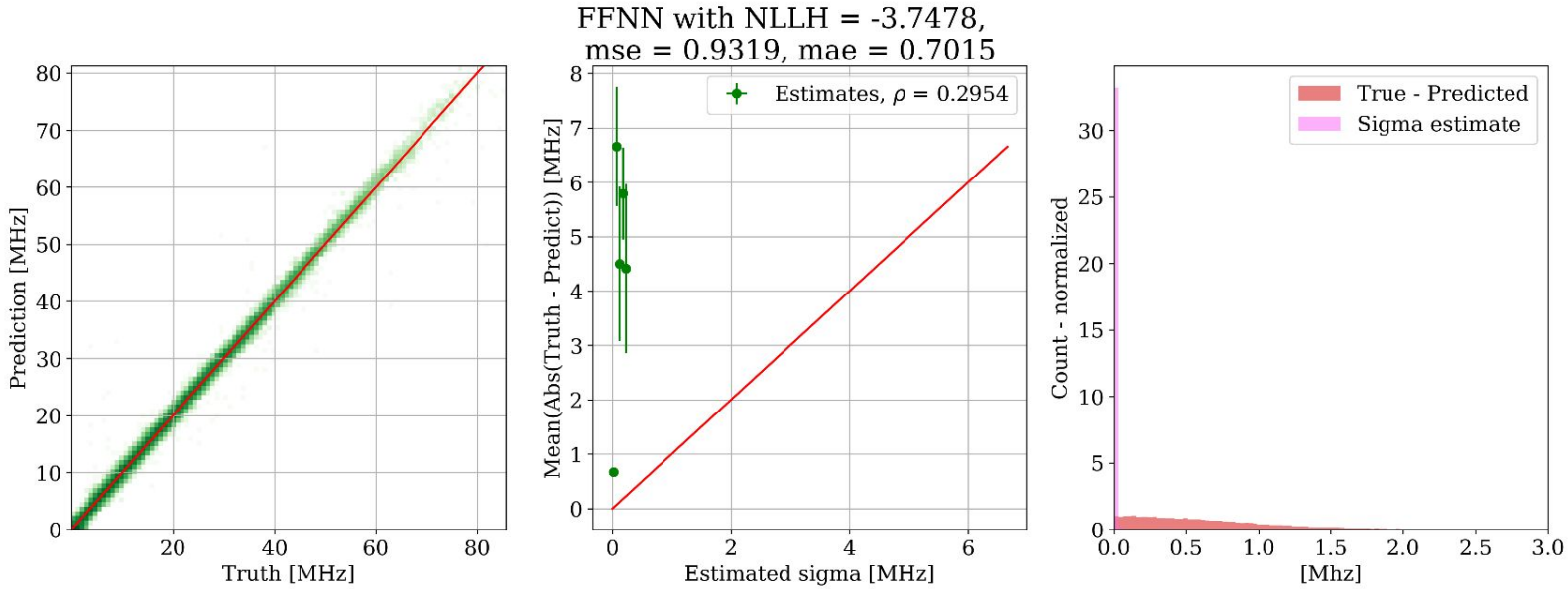
LightGBM

FFNN

Standard implementation and simple models. Simple is good for the constraints mentioned

Feed-Forward Neural Network (regular kind): First results

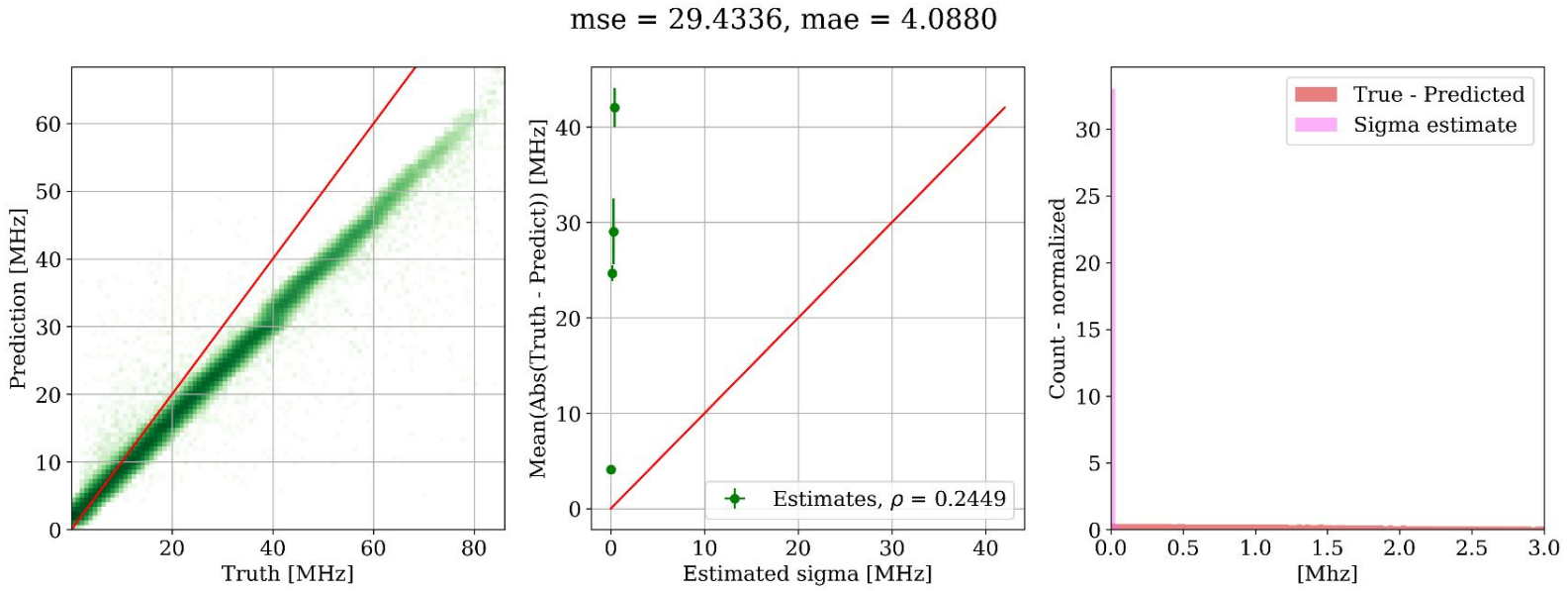
- 4 layers with 50 nodes each
- ~13000 parameters (OPX compatible)



For specifics on the FFNN, see appendix

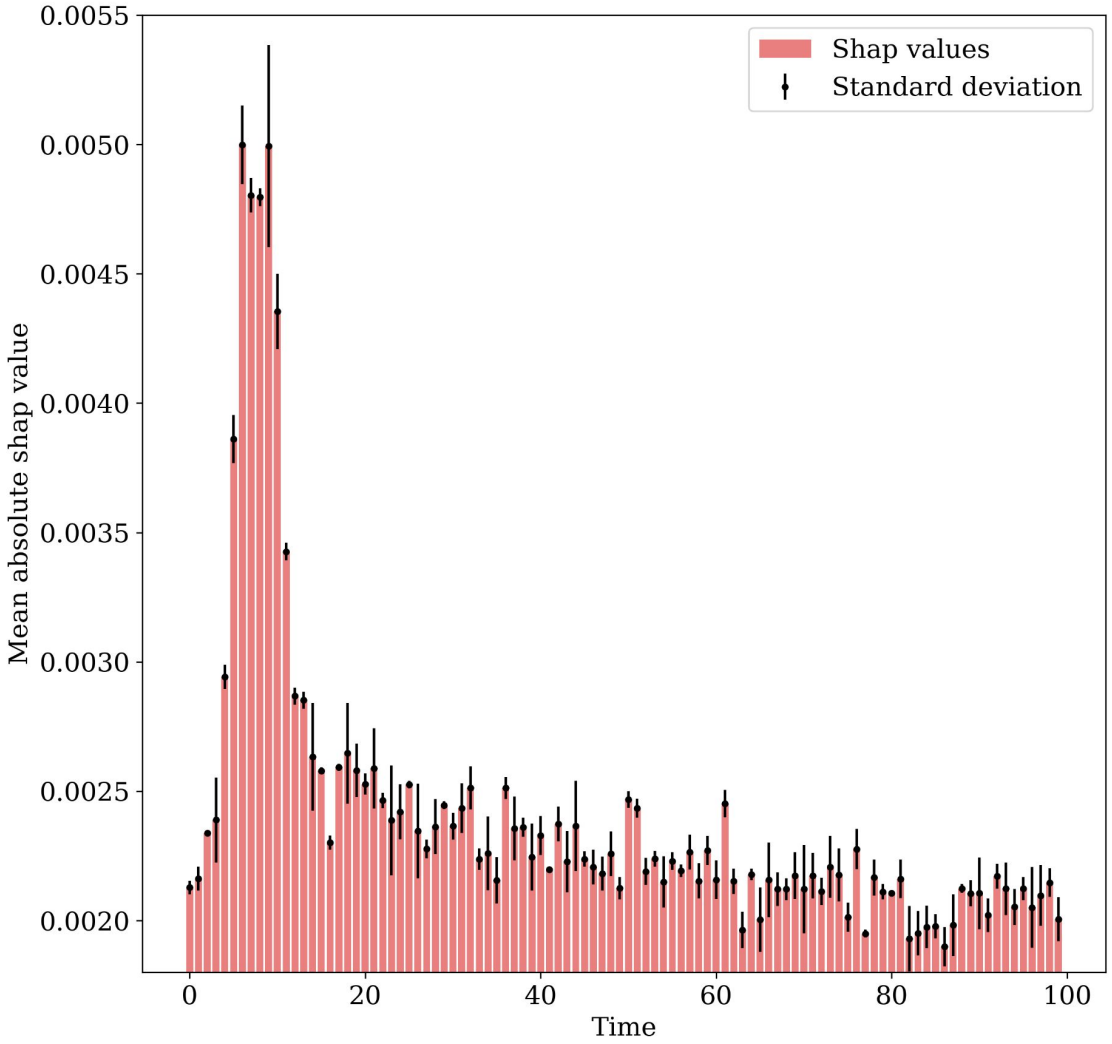
Feed-Forward Neural Network (trained on padded data):

- 4 layers of 100, then one of 32 and 10
- ~43000 parameters (borderline OPX compatible)



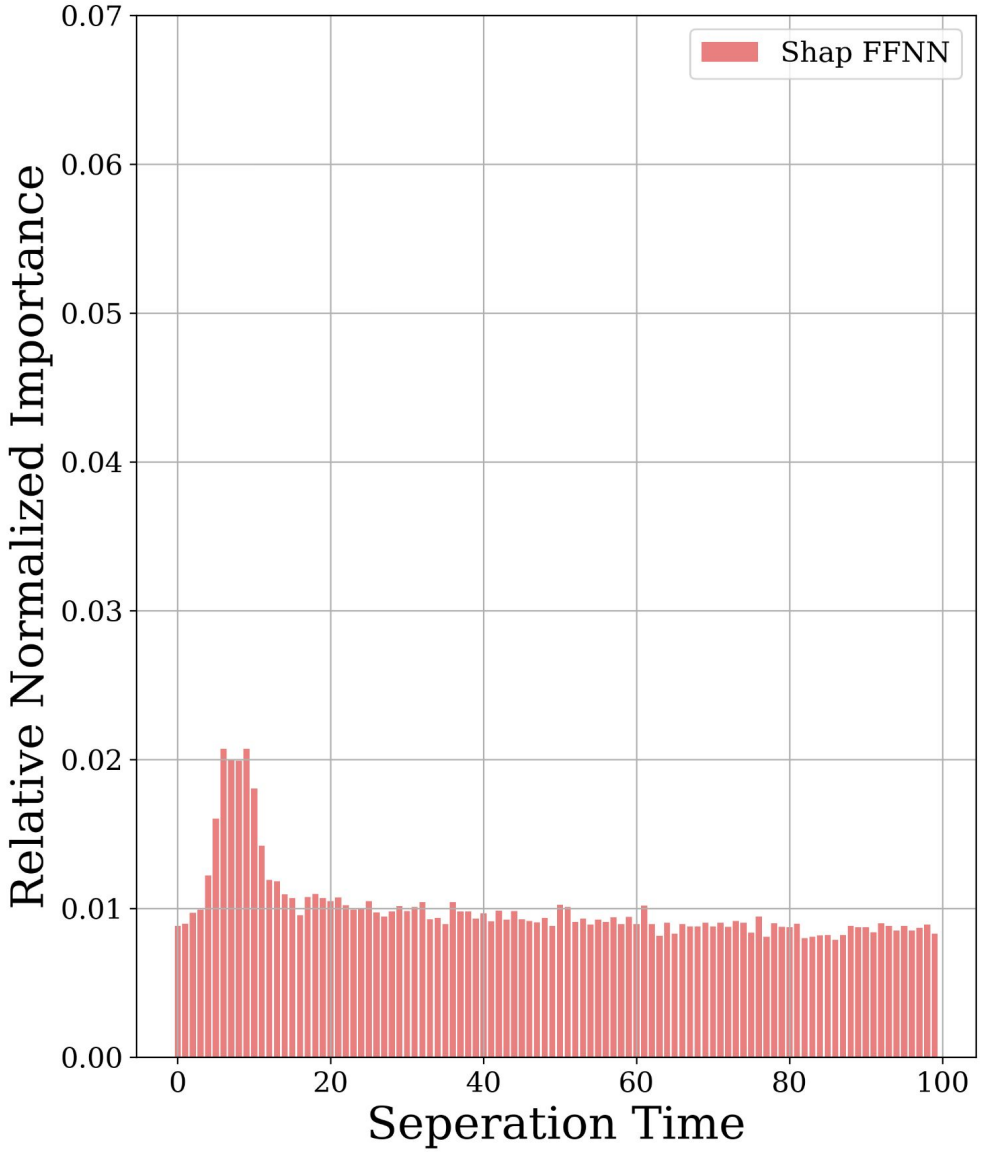
For specifics on the FFNN, see appendix

SHAP analysis

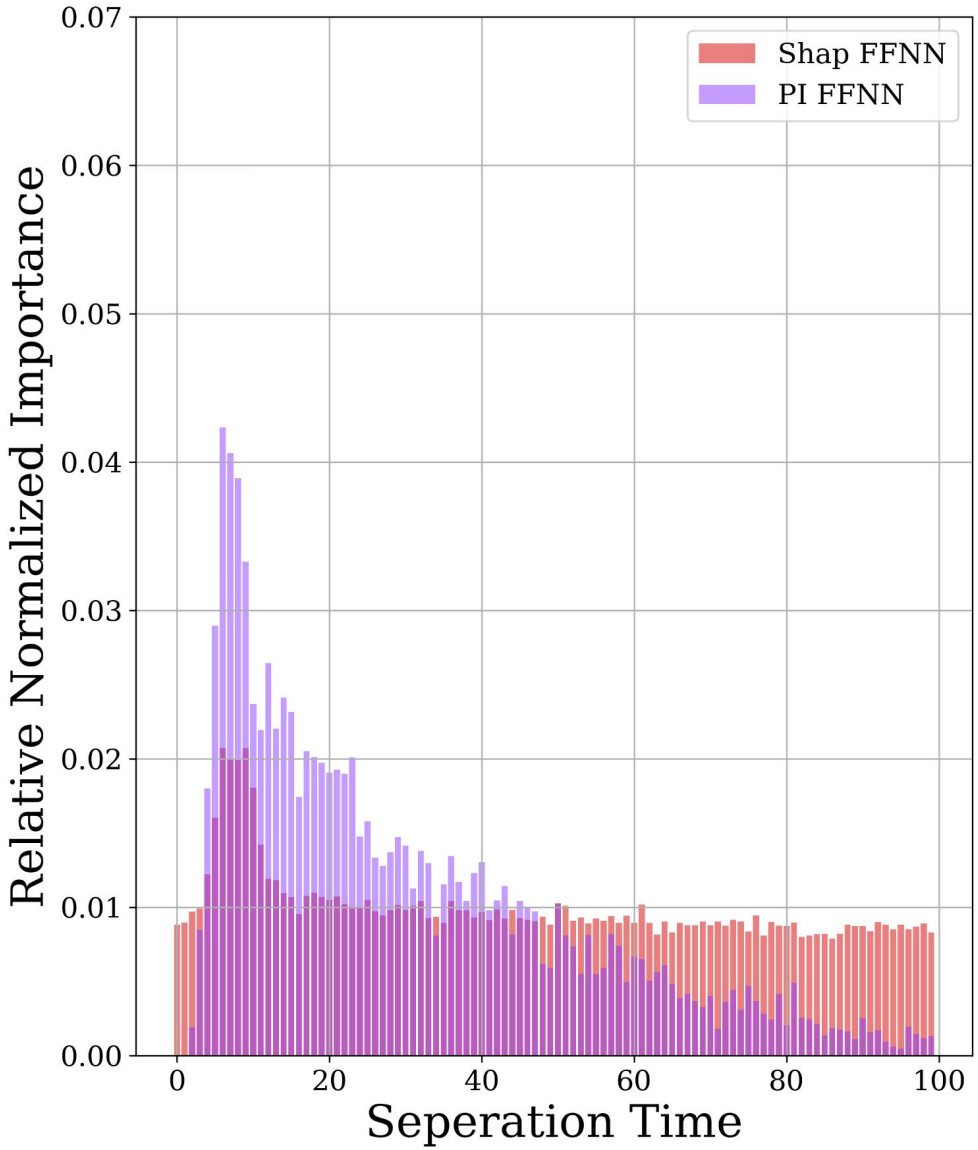


Z_{12}=0.12

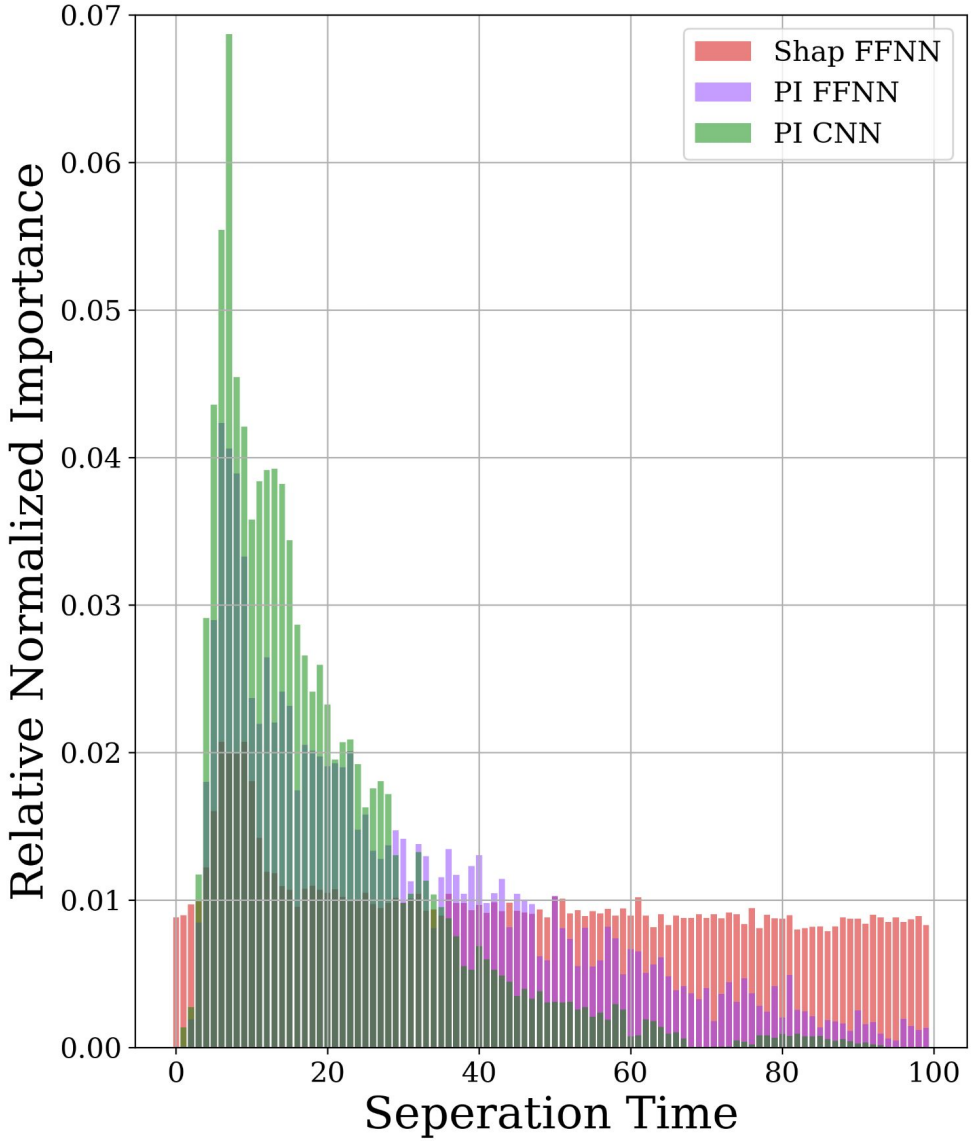
Permutation analysis



Permutation analysis

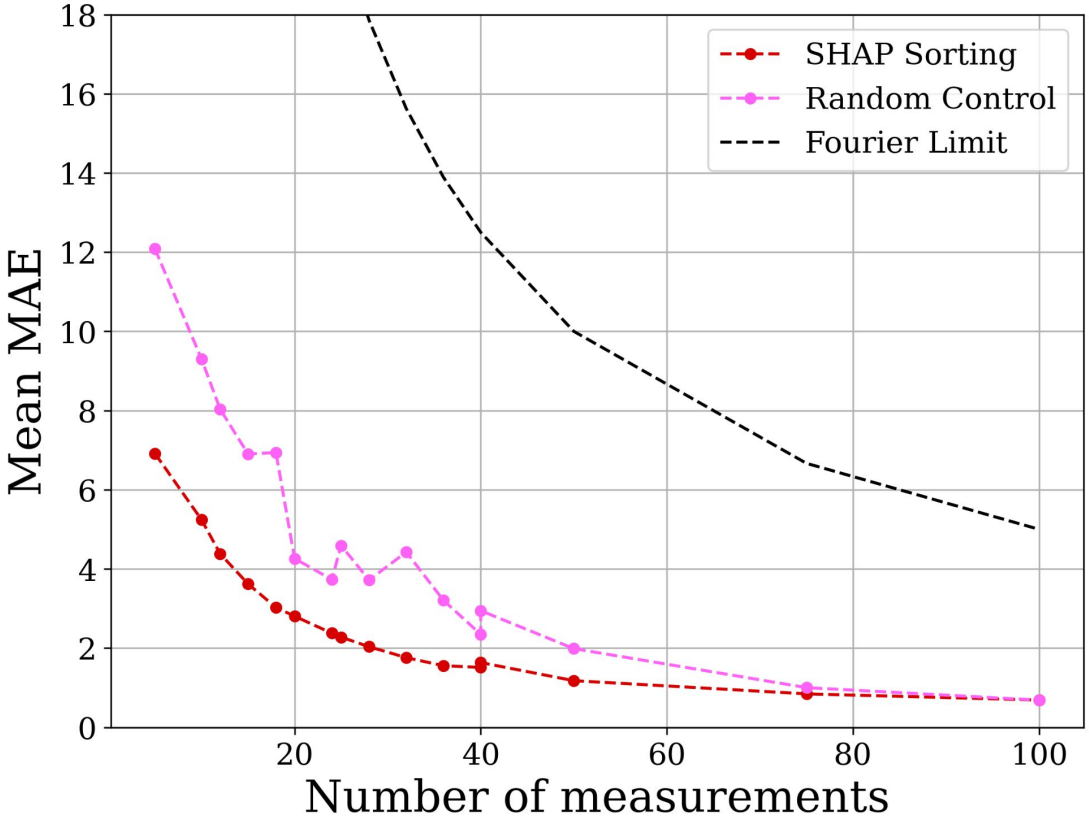


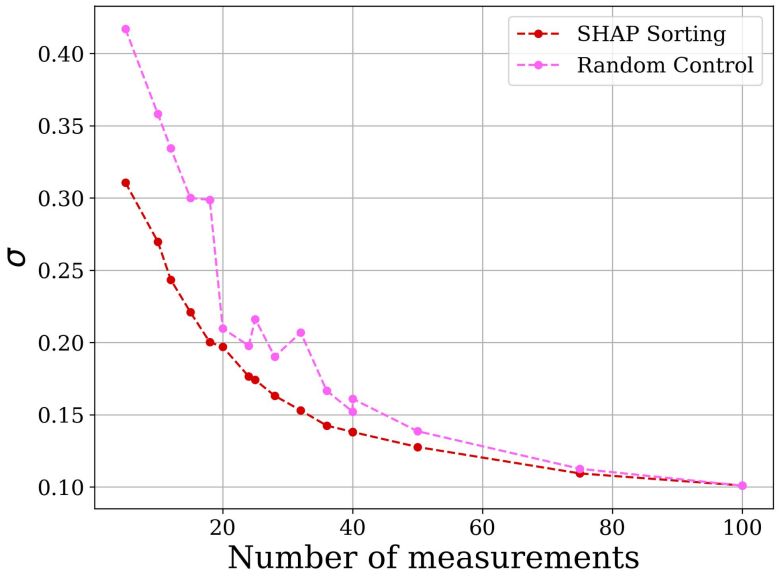
Permutation analysis



Reducing data: Retraining

- Remove low SHAP times from dataset
- Retrain model

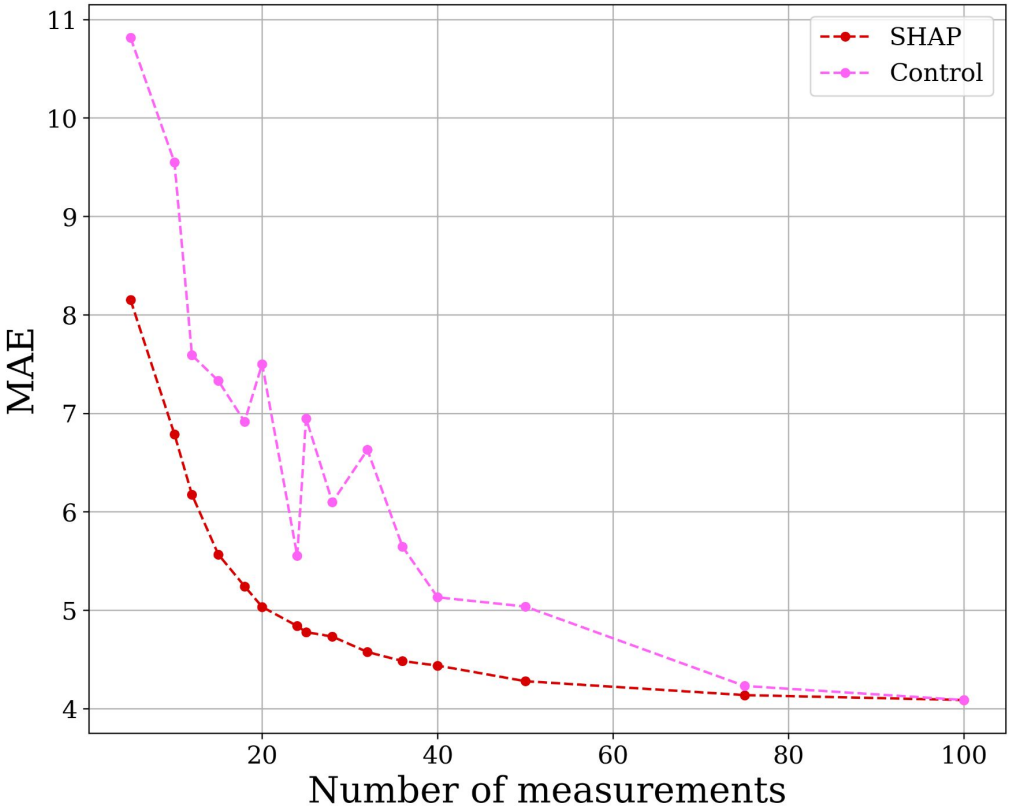


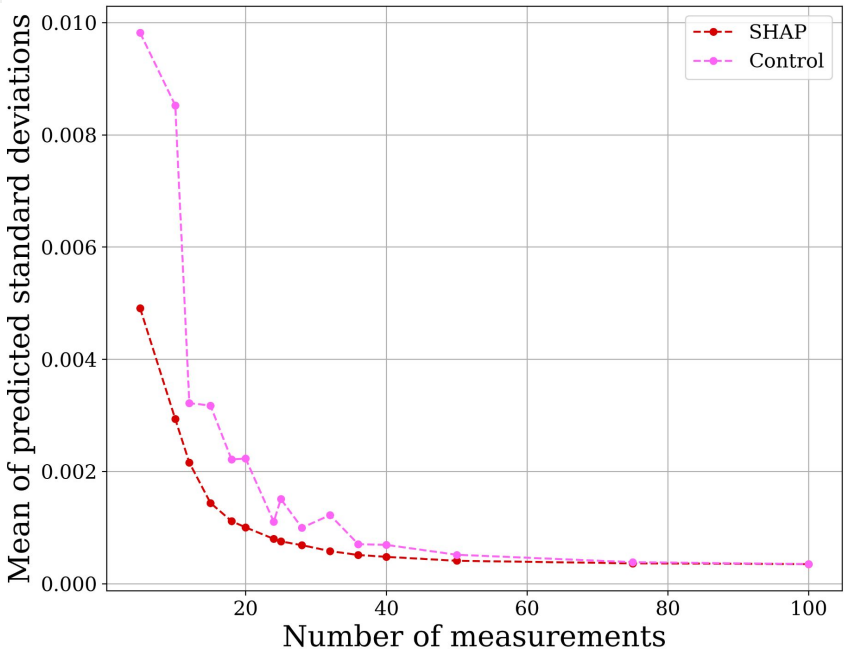


- Sigma highly correlated with true error
- Beats fourier resolution
- Potential to use more measurements at 30 measurements

Reducing Data: Zero-Padding

- 4x100+32+10 nodes
- Model trained uniformly random zero-padded data

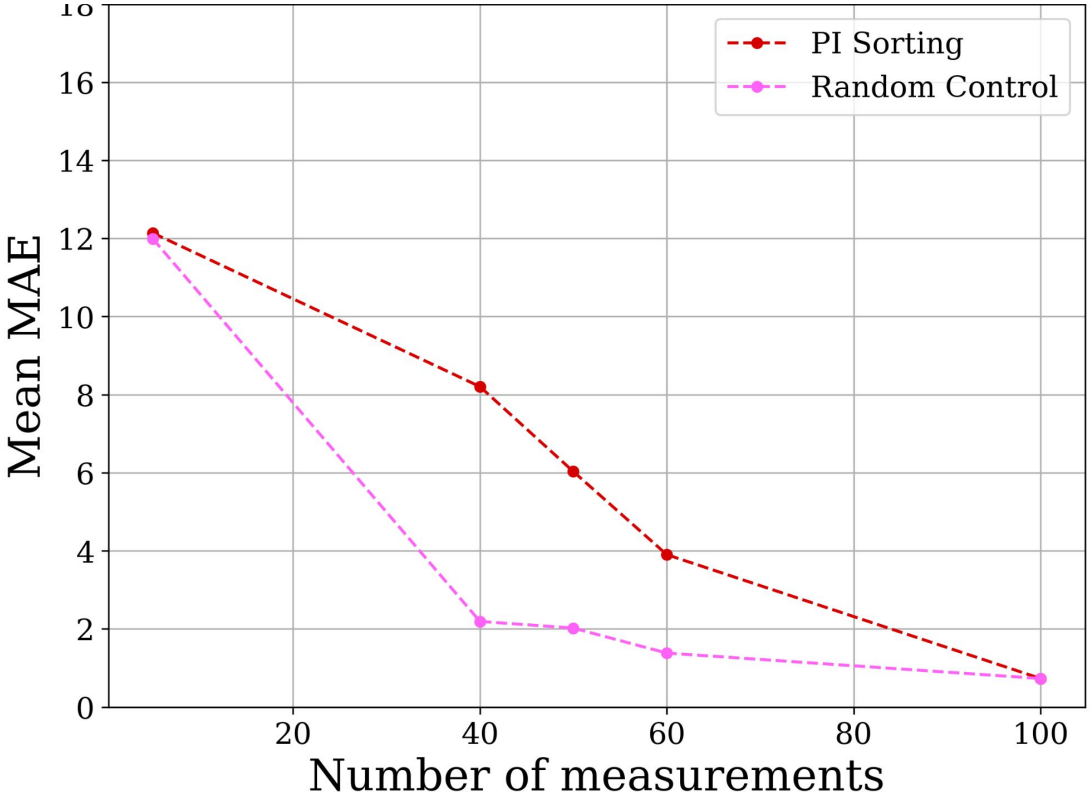


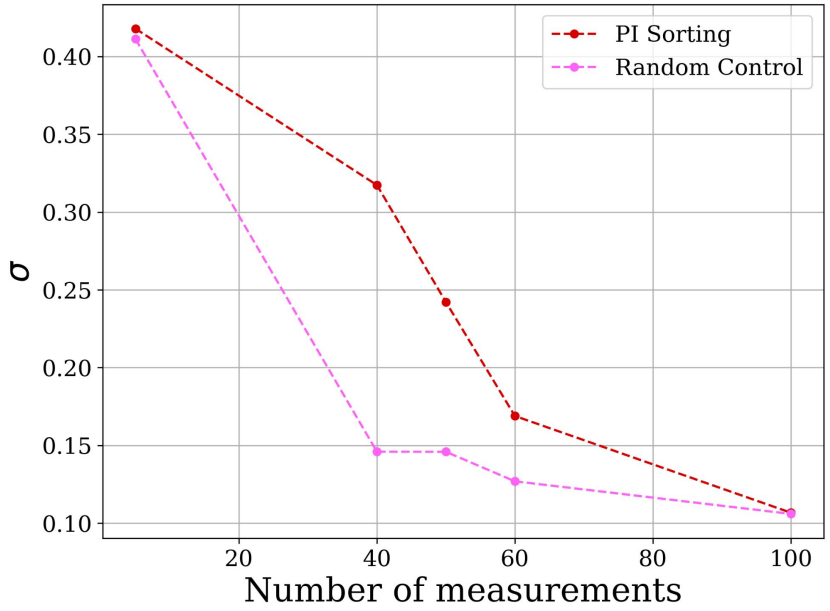


- One model for all feature constellations
- Same features as before, but worse MAE
- Monotonically convergent

Reducing Data: Permutation Importance

- 4x50 nodes
- Does PI give better MAE?





- Maybe not

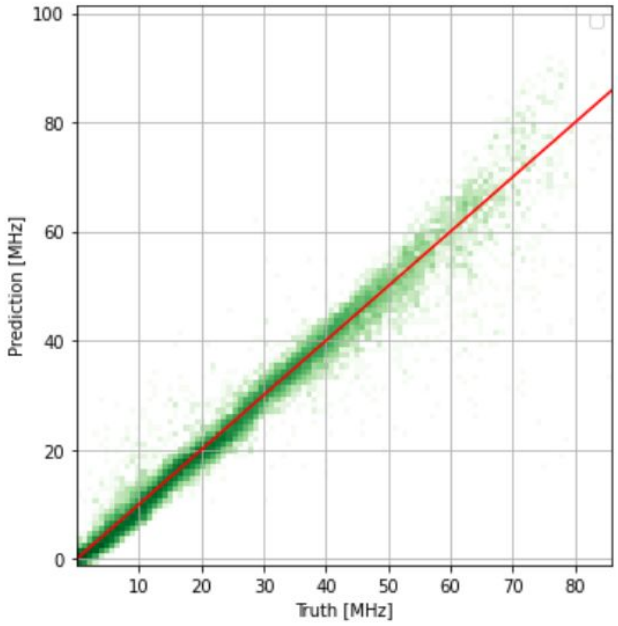
- Likely explanation: PI overestimates length of peak and underestimates correlation in it. More than 10 points in peak won't give better results.
- Still better than Fourier

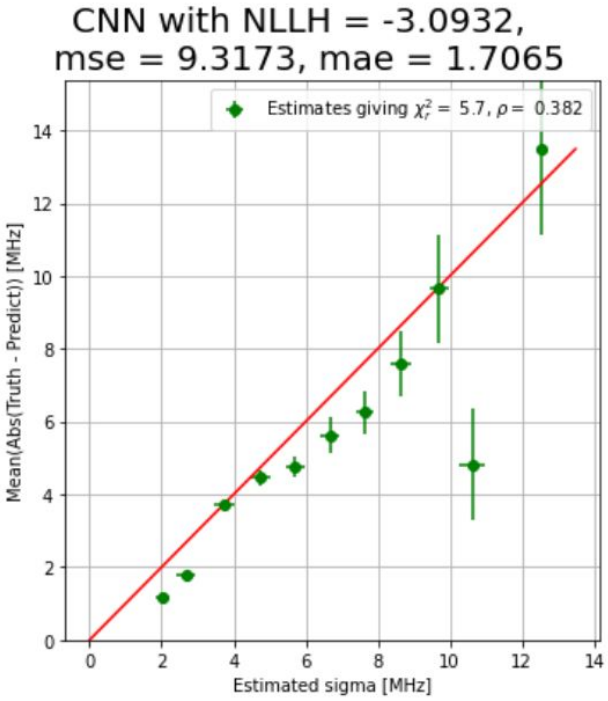
CNN

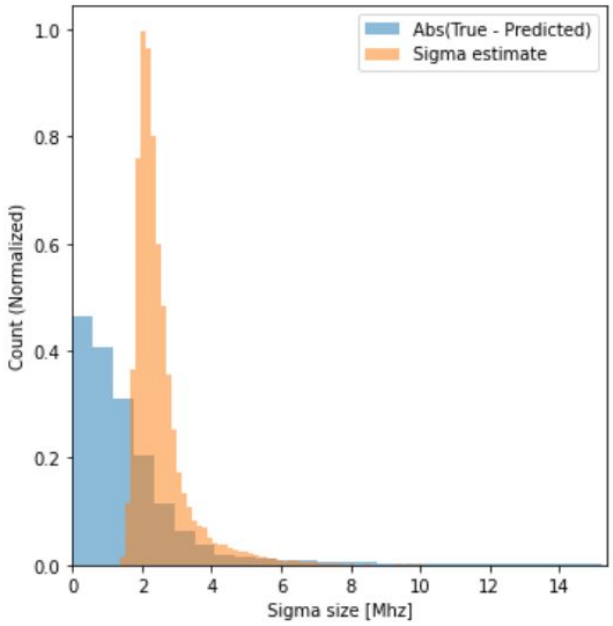
A Convolutional Neural Network should perform at least as good as the FFNN

- Works with convolutions
- Input just ones and zeros - includes time

First results on validation dataset

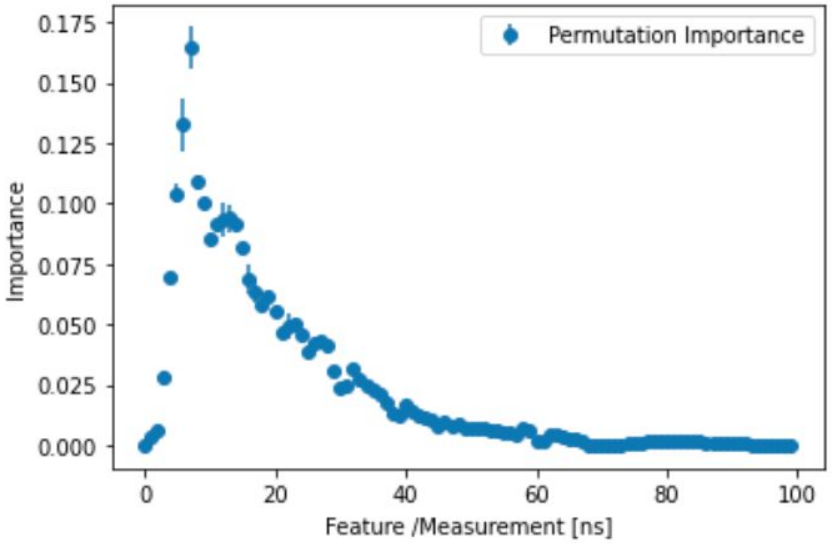


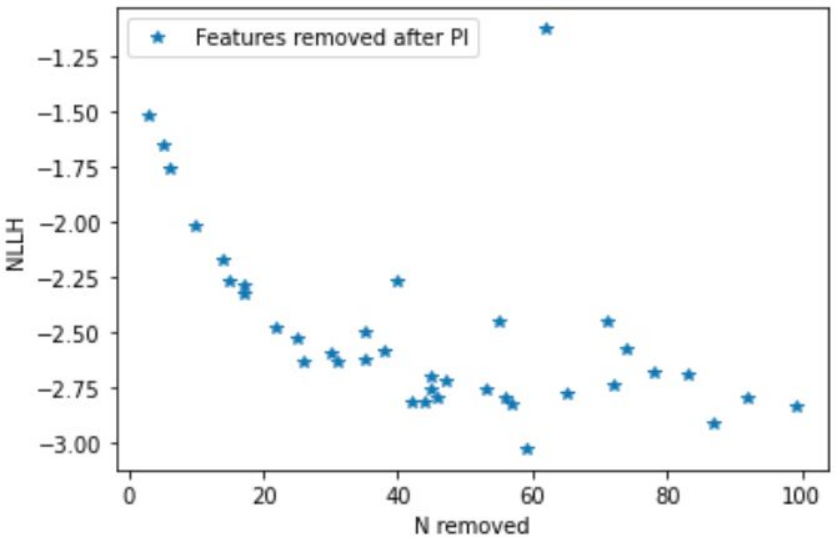




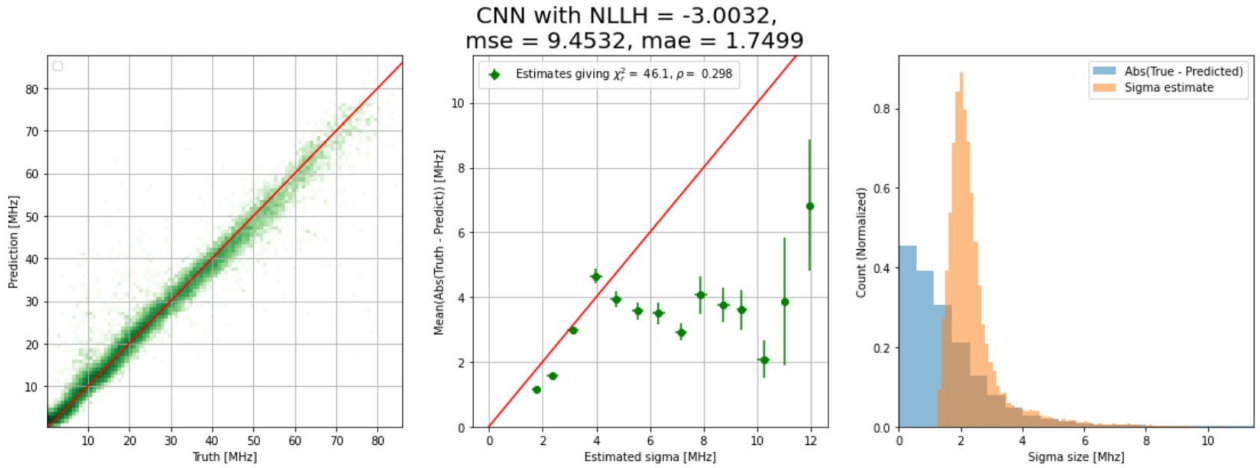
For stats on the plot above, se Appendix, slide ???

Results with reduced data - 50 Features





The plot to the right is made with the same setup as the previous slide

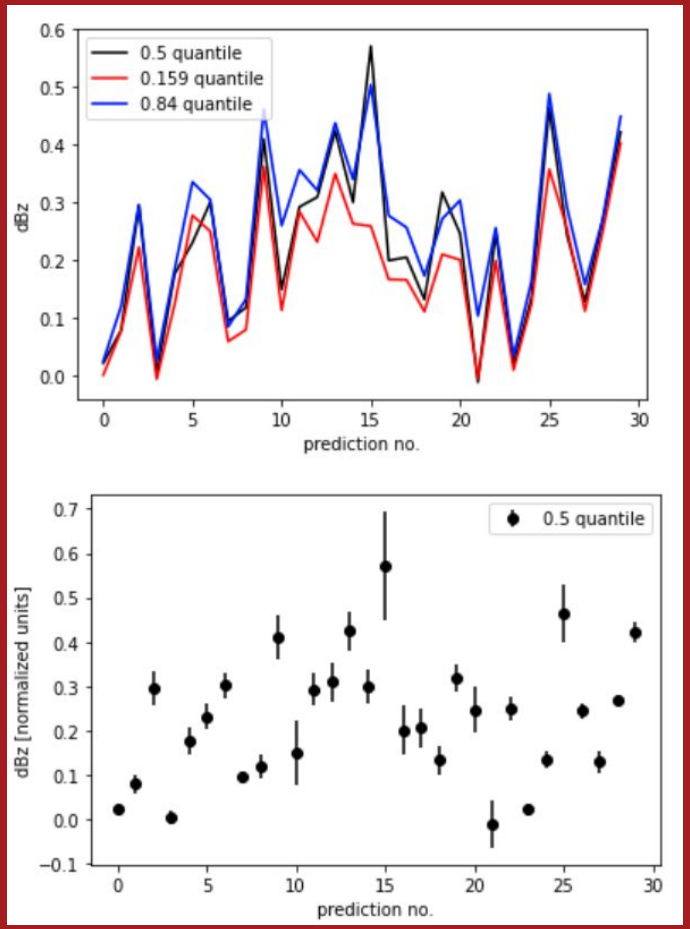


LightGBM

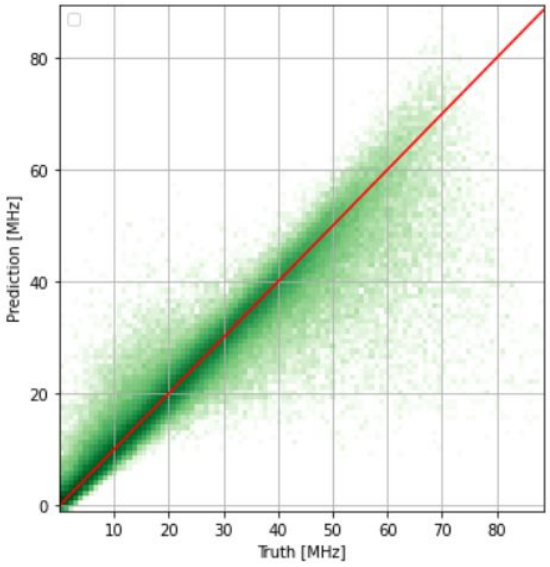
A tree based network → *Might* be implementable on the FPGA

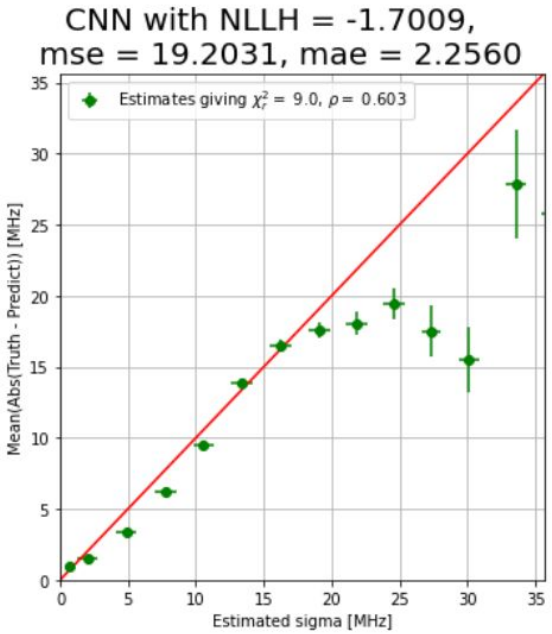
Opportunity:
Might be smaller and faster

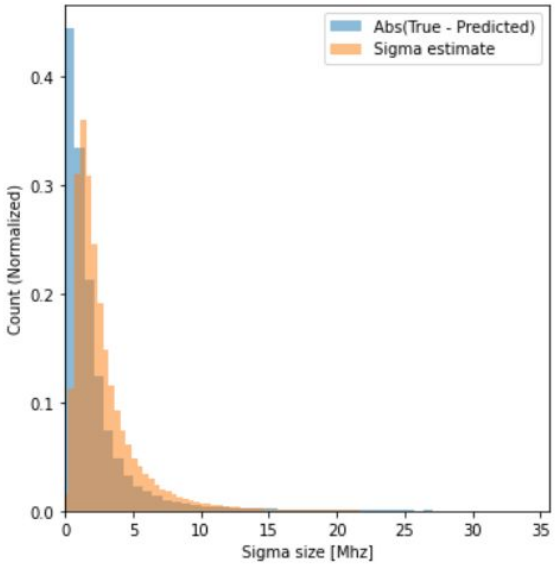
Problem:
No built-in NLLH → We use quantile metric



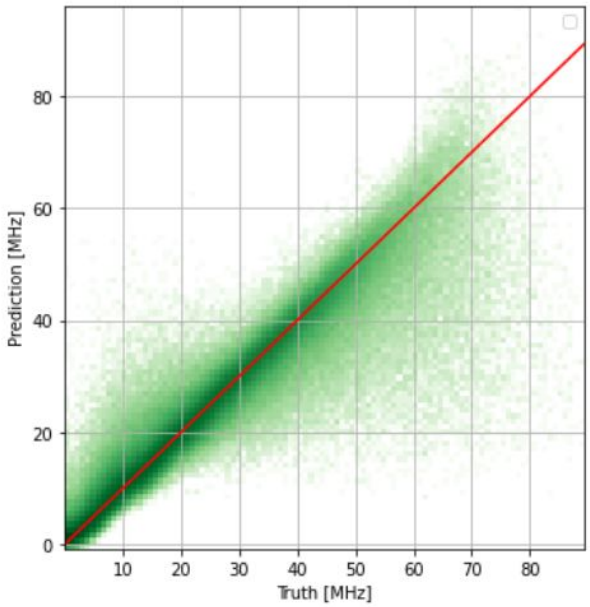
Initial results using standard LightGBM

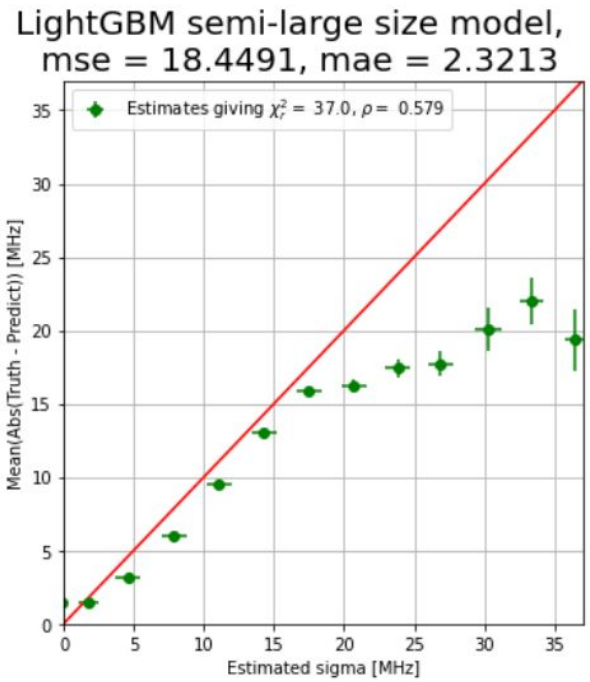


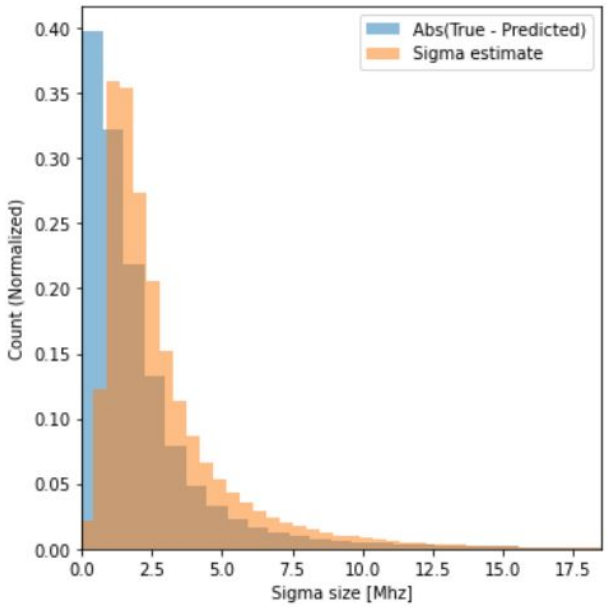




Results with reduced data







GAN

Live Next Separation Time Prediction

Almost works

Bug in keras.utils.to_categorical and possibly also in tf.Tensor.make_ndarray:

```
NotImplementedError: Cannot convert a symbolic tf.Tensor (RegressorLoss/Floor:0) to a numpy array. This error may indicate that you're trying to pass a Tensor to a NumPy call, which is not supported.
```

```
AttributeError: 'Tensor' object has no attribute 'tensor_shape'
```

Structure (will be improved before presentation with graphs)

Regressor outputs regression and sigma from zero-padded data

Zero padded data and regressor output given to classifier, that outputs index (separation time from 1 to 100 [ns])

Old zero padded data plus measurement from suggested index given to regressor, outputs new sigma, used as loss for training, and convergence criterion during actual use

In the works

Live Estimating Next Ideal Measurement Time
- Code structure done

Combining measurements at identical separation times

Including experimental lower level data such as measured voltage (instead of binary classification)

Grand Table of Results

Setup	Mae - All [MHz]	Mae - Reduced [MHz]	Model size [Total Params]	Error correlation [Pearson]
FFNN (4x20)	0,778	1,29	**3.642**	0,373
FFNN (4x50):	**0,701**	**1,18**	13.602	0,295
FFNN (4x100+32+10):	4,08	4,58	45.712	0,245
CNN:	1,71	1,75	128.130	0,382
LGBM (large):	2,26	2,32	<33.000	**0,603**

Outro

Evaluation of the various models:

FFNN

- Different down-sampling methods are viable for different purposes.
- Difficult to get the uncertainties to converge properly

CNN

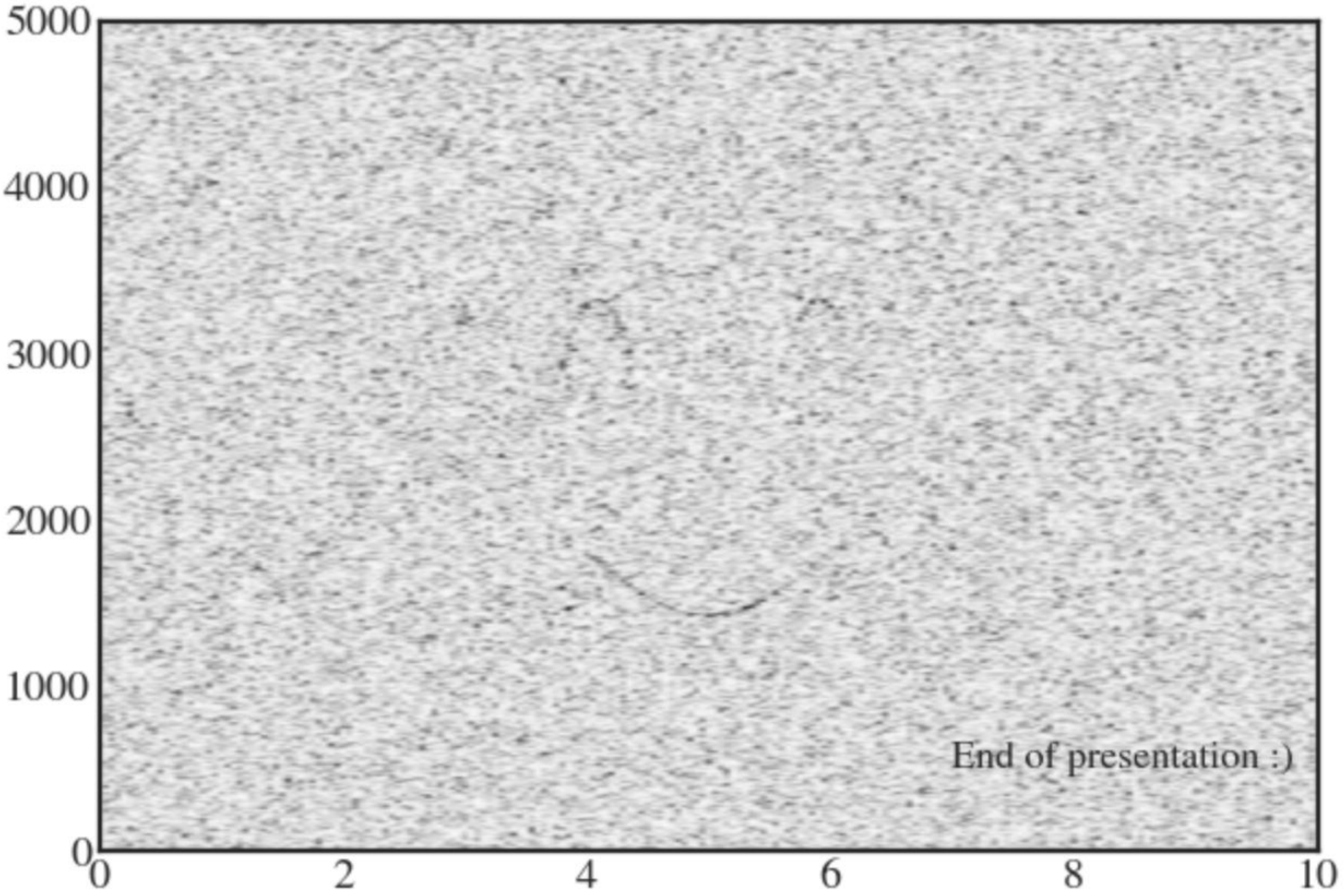
- Inherent time-series data maintained
- Large amount of parameters
- Good predictions and good uncertainties

LightGBM

- Good predictions and very good uncertainties. Medium amount of maximum parameters

Three-step plan:

1. ✅
2. ✅
3. 50%


5000
4000
3000
2000
1000
0
0
2
4
6
8
10
End of presentation :)

AppendiX

Outline:

Data Processing
- Data description [RKU]
- Simulator Descr. [RKU]
- Preprocessing Descr. [OLI]

Implementation Constraints [OLI]

NLLH-Loss [OLI]
- Motivation
- Implementation

FFNN [OLI]
- Model description
- Hyperparameters

Downsampling [AJSN]
- Reshaping vs. Zero-padding
- SHAP/PI
- N_meas vs. MAE
- FFNN robustly trained on random
  - Hyperparameters

CNN [AESD]
- Model Description
- Hyperparameters

LGBM [AESD]
- Model description
- Hyperparameters

GAN [RKU]
- Motivation
- Regressor Loss Implementation
- Training Implementation

Data processing

Data description

Simulator description

Preprocessing description

What are we working with: The Quantum

The Quantum

- Nuclear spin In GaAs Dots is hot at 30 mK
  - 10^4 to10^5 fluctuating nuclear spins
- Singlet state of 2 electrons in 1 dot
- Separated for a time to 2 dots (Separation Time)
  - => Magnetic field gradient between electrons
- Attempt recombination of electrons to single dot
  - Only successful if electron pair in singlet state
  - => Boolean (Pauli blockade/no Pauli blockade)

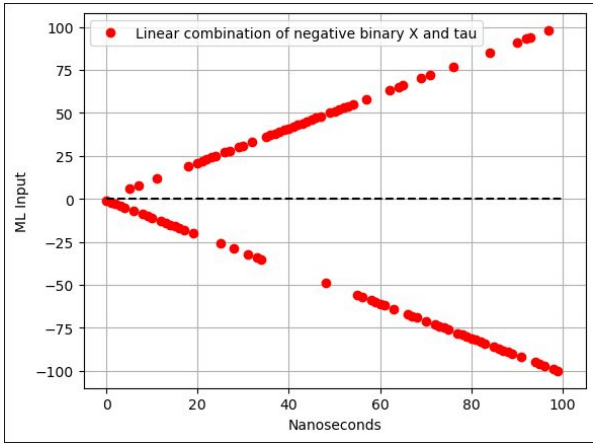
We simulate this using simulator provided by expert (modified to be faster)

Simulator Description

The simulator we received from the QDev group, works by firstly producing an array of the true dBz values (the frequencies we're trying to predict). It then applies a Metropolis-Hastings algorithm, which also accounts for various biases in the system, such as other residual quantum effects, and it provides the zebra plots that we have shown above. The full output data that the simulator provides is of the form: (2, 100, runs), where runs is an arbitrary number, which is simply the amount of data we can train on. The explanation behind this data structure is: The initial dimension of 2, is the binary value vector of 0 and 1, and the separation time vector, both of which is 100 long.

Preprocessing description

Due to the limited number of numbers that can be represented simultaneously on the OPX (64000 numbers), we sought a way to minimise the amount of input required for the neural network. This was achieved by taking the 100 column of binary values (0, 1), transforming the 0s to -1s. Then taking the 100 column of separation times (1ns, 100ns), and multiplying it with the new negative binary column, thusly producing an input which looks as follows:
The result is that we have halved the amount of input data, and that it should make the model robust against zero- padding the input data.



Implementation constraints

As mentioned before, the OPX has limited memory, so the size of the neural network is of crucial importance. However, the OPX also has a limited number of math operations it can do, meaning that all the operations performed in the network, need to implemented via these limitations. Since the input data is of the form (-, +), we would have liked to use an activation function, which is symmetric around 0, like tanh(). We ended up using the Relu and Elu activation functions, both of which are a native function on the OPX. Due to the same reasons, Fourier transforms are also not possible on the OPX, meaning a CNN is not the best candidate for this specific project.

Implementation constraints

However, it was still implemented as a benchmarking model, and to see if implementing FFTs in future models of the OPX would be favourable.
The data we were provided, was of the form of a time-series, meaning the data has intrinsic linking based on the position in the array. This was another constraint, due to the fact generally, a FFNN cannot see this correlation. This was another reason for performing the linear transformation described earlier; as a safeguard against losing the inherent information in the positional component of the data.

Negative Log Likelihood Loss

Motivation

Implementation

Motivation for NLLH Loss

The motivation for a negative log likelihood Loss functions, is fueled by the desire to construct a GAN, which would be able to, while the experiment is running, predict the next optimal separation time to perform a measurement at. All training has been performed on simulated data, whereby the true label is known, however, while performing the experiment, the true label is not know, so we need a live way of measuring the error on the estimation. The (negative) log likelihood loss function provides just that, as it outputs the uncertainty.

$$NLLH = -\frac{(y_{true} - \mu)^2}{2\sigma^2} + \frac{1}{2}\log\left(\frac{1}{\sigma^2}\right)$$

Implementation of NLLH loss:

We wrote a custom function, which contains the math that can be seen below. An important note, however, is that in the actual code, the parameter it estimates is actually σ^{-2} as this performed better and converged more quickly. This custom function was then utilised in the various models we used, generally the keras framework was utilised, as the loss function is should optimise for.

$$NLLH = -\frac{(y_{true} - \mu)^2}{2\sigma^2} + \frac{1}{2}\log\left(\frac{1}{\sigma^2}\right)$$

Feed Forward Neural Network

Model Description

Hyper Parameters

Model description of the Feed Forward Neural Network

The FFNN is a standard neutral network, in that you have an input layer, with the dimension of your input array. Following this, you have an indiscriminate number of dense layers with an optimised number of nodes in each layer. Generally, this type of network is not able to “understand” a time-series data-set, as it does not necessarily see the link between the positional component of the data in the array. But by feeding it the linearly transformed data as described earlier, we hope to see the linkage remain in the output.

Model description of the Feed Forward Neural Network

These models did indeed prove more competent than expected given the constraints on their size and complexity. The NLL estimator was a partial succes since they often got stuck in underestimating their own errors. They did however show some correlation and by personal investigation we were able to conclude that the data looked qualitatively fine if errors were multiplied by roughly 80.

Model description of the Feed Forward Neural Network

The error correlation (Pearson coefficient between MAE and estimated sigma) was however fairly weak at around 0.3 and it might therefore be smart to work further on refining this error estimation.
The loss function was set to estimate sigma^-2 since this was thought more robust, than estimating very small floating point numbers

Hyper Parameters of the FFNN

All the FFNNs have the following constant hyper parameters:

Learning rate	Loss Function	Activation function	Optimiser	Batch-Size	Batch-Norm momentum	Early stopping patience
0.01	Custom NLLH	Relu	Adam	2^8	0.9	15

The only thing that varied between models, was the size of the network.
We had the following sizes:

4x20	4x50	4x50	4x100 + 32 + 10

SHAP analysis

To find out which time measurements are most important, we do a SHAP analysis on random samples of data for the 4x20 FFNN model. The analysis is performed on 3 subsets sampled from a large dataset of 3 million and a smaller of 200000 overhauser gradients. The Z-test is between the two first of the subsets (from separate datasets) and is considered sufficiently small at z=0.12.

After concluding that the different SHAP-sets are similar, an average is taken and error bars extracted from the variation for each point. This is the SHAP values used for further downsampling (Except where permutation importance is used).

SHAP vs Permutation

Since permutation importance is more computationally efficient, we wanted to check whether it agreed with the SHAP values, so that we could train on it and also to cross-validate the SHAP method.

The main analysis was performed on 200.000 samples from the dataset with 3 million in total and this was then compared with the average SHAP array.

We also used PI analysis from the CNN (though an older version of the CNN) and were able to conclude that all methods agree on roughly the same peak and only disagree on the slope after the peak.

SHAP vs Permutation

It While it was difficult to determine exactly who was right, we mostly stuck with the SHAP values for the FFNN models since we thought the method would be more robust. Given more time we would have trained on the top features of the PI method to see if it reduced our MAE more efficiently.

Downsampling, Padding or retraining

The motivation to downsample was clear. If fewer measurements are sufficient, it allows for longer gate times in the quantum computer. Shorter times also mean we have fewer errors from dephasing and drift of the magnetic field and it might therefore be worth it to sacrifice some accuracy for shorter calculation times.

Finally, since measurement time is the main constraint for the experiment, it might be possible to collect multiple samples at the same separation times and improve menasurments while making them smaller

Downsampling: Retraining

To do this we came up with two different methods. The most straightforward was simply to remove the least important features from our data-sets, retrain the model and see how high the accuracy would remain.

The advantage of this approach is that it is likely to perform as well as possible, given the method and the data. However, it presents a challenge for extension to uneven sampling done “live” (ie. while experiment is running).

Therefore we also came up with a strategy which could work at a wider range of measurements.

Downsampling: Padding

This strategy consisted in using a model trained with 100 inputs, but where measurements would be removed by inserting zeros instead. This model was made as large as could be feasibly assumed possible to run and was then trained on a “staircase” of zero-padded data. The full training set consisted in 3 million data points where the first 3e4 entries were left as is, the second 3e4 entries had 1 time measurement changed to a zero at random (ie. lost one one hundredth of their measurements). The next 3e4 entries then had 2 measurements padded until the final 3e4 entries were left with only 1 non-zero measurement. In this way, the network was exposed to a wide range of missing data.

Downsampling: Padding

As expected, the poor quality of training data made the network generally worse, but it also proved to be fairly robust when exposed to different zero-padded datasets. As is seen when we run this model on a range of padding, the change between 20 measurements and 80 is very small with this network. While the MAE is still worse than the retrained models, further improvement has much greater potential for this model, since it could work in conjunction with another machine learning algorithm which would predict what new data points to add. This is discussed in our GAN section

Honourable dead: Approaches that didn't make the cut

- Bayesian estimation, very hard algorithm to implement
- FFT: not refined enough to resolve the frequencies, especially not with the added quantum noise
- RNN: Very difficult to get it to converge

CNN

Here comes the model stats: We gave it 100.000 samples, 12 epochs, learning rate 1e-3. Best parameters found with Optuna.

```
Model: "sequential_278"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_996 (Conv1D)         (None, 32, 96)            672

 conv1d_997 (Conv1D)         (None, 16, 128)           36992

 conv1d_998 (Conv1D)         (None, 8, 64)             41024

 flatten_275 (Flatten)       (None, 512)               0

 dense_550 (Dense)           (None, 96)                49248

 dense_551 (Dense)           (None, 2)                 194

=================================================================
Total params: 128,130
Trainable params: 128,130
Non-trainable params: 0
_________________________________________________________________
```

Hyper Parameters for LightGBM

Number of samples used 700.000

Loss function				
N Layers				
N Nodes				
Activation				
Optimiser				
Batch-size				
Batch Norm Momentum				
Learning rate				

Bibliography

A Machine Learning Approach to Bayesian Parameter Estimation - Nolan, S., Smerzi, A., Pezzè, L. NPJ, Quantum Information https://doi.org/10.1038/s41534-021-00497-w

Suppressing Qubit Dephasing Using Real-Time Hamiltonian Estimation - Shulman, M.D., Harvey, S.P., Nichol, J.M., Bartlett, S.D., Doherty, A.C., Umansky, V., Yacoby, A. Nature Communications DOI: 10.1038/ncomms6156

https://stats.stackexchange.com/questions/564809/using-a-neural-network-to-learn-linear-regression-variance/564811#564811

D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Orlando, FL, USA, 1994, pp. 55-60 vol.1, doi: 10.1109/ICNN.1994.374138.