MACHINE LEARNING-ASSISTED CALIBRATION OF THE ACCELERATOR SIMULATOR HPSim

C. Leon, A. Scheinker, P. M. Anisimov
Los Alamos National Laboratory, Los Alamos, United States

Abstract

Manually calibrating the HPSim simulator to the LANSCE accelerator is time-intensive and demands substantial domain expertise. In this work, we investigate the use of machine learning (ML) to automate much of the calibration process and substantially reduce tuning time. Specifically, our focus is the calibration of the front-end of the accelerator, which involves obtaining the amplitudes and phases of the pre-buncher, main buncher and tank 1 of the drift tube linac (DTL). To get empirical data of the accelerator, we use current-phase curves obtained from absorber/collectors, both with the pre-buncher on and off. We derived features from the curves (e.g., standard deviation of each signal and average distance between them) and used these features to train the ML models. By combining classical ML methods—gradient-boosted decision trees and Random Forests—with a state-of-the-art transformer model, we achieve a significant speed-up of the calibration process, from about a month of human expert labor to 2–3 days of mostly computational processing.

INTRODUCTION

The LANSCE (Los Alamos Neutron Science Center) facility accelerates both positive and negative hydrogen ions, H^+ and H^-, respectively. Due to the enormous complexity of the machine, the intricate physics (e.g. space charge effects), and the limited amount of non-invasive diagnostics available, much of the analysis is done on the accelerator simulator HPSim [1]. Over time, the state of the accelerator drifts with normal wear-and-tear, age of the facility, equipment replacement, etc. As a result, HPSim periodically needs to be re-calibrated to reflect the current state of the machine.

Specifically, the amplitudes and phases of different components need to be tuned. For the front-end, we are concerned with three components: the pre-buncher (PB), the main buncher (MB) and Tank 1 of the DTL (T1) (see Fig. 1). An absorber/collector was placed behind tank 2, which was shut off. This produces a current vs. phase plot. The phase is with regard to the DTL nominal frequency of 201.25 MHz. Using the measurements from there, HPSim can be calibrated for the front-end of the accelerator. Specifically, we aim to determine five physical variables: the amplitude of T1 (T1Amp), the MB amplitude (MBAmp), main buncher phase (MBPhs), pre-buncher amplitude (PBAmp) and the pre-buncher phase (PBPhs). By convention, all phase variables are taken relative to tank 1.

There are many difficulties tuning PB, MB and T1 [2]. Manually calibrating HPSim is a laborious process and requires significant expertise. A brute-force approach is infeasible. To see this, based on the required resolution (see Table 1), approximately 5×10^{13} states would need to be evaluated. Using a high performance computing platform, we can currently run $\sim 5 \times 10^3$ simulations per day. Thus, an exhaustive search would require roughly 27 million years to complete.

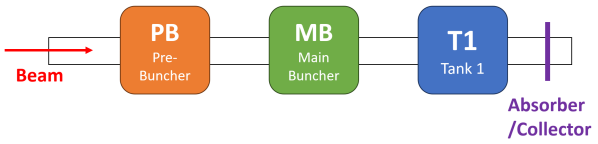

Figure 1: Simplified schematic showing the front-end of LANSCE and the relevant components.

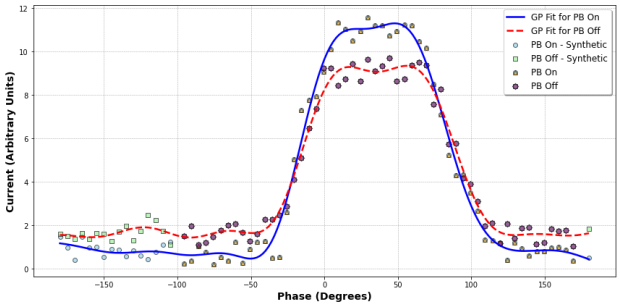

Figure 2: An example showing the raw data from the absorber/collector, the synthetic data created for the tails and mean functions from the Gaussian Process fits.

Hence, we turned to machine learning (ML) to speed the process up while making much of it automated. Specifically, we are interested in the inverse task of given experimental parameters, determine the physical variables of the accelerator components.

DATA

Table 1: Parameter Ranges and Resolution Requirements

Parameter	Range	Accuracy	Points
MBAmp	10–12	0.01	200
MBPhs	−180° to 180°	0.1 °	3600
PBAmp	3–5	0.01	200
PBPhs	−180° to 180°	0.1 °	3600
T1Amp	25–26	0.01	100

LANSCE Data The real world data from LANSCE. Two current vs. phase signals are obtained from an absorber/collector pair, one with the PB on and one with the PB off. For all that follows, only the H^+ beam will be used.

Using the raw signal comes with many issues. First, not all signals are sampled at the exact same phases. For exam-

ple, for one measurement the data might be collected at 68.2 degrees and the next signal is sampled at the nearby 67.9 degrees. Second, there is significant noise. To be able to arbitrarily sample for training models and extract a noise reduced signal, we used a Gaussian process (GP). We modeled the signal as consisting of pointwise noise and correlations between different phase values. Specifically, we used both a radial basis function and white noise for the kernel. As an added bonus, uncertainty quantification can easily be done with GPs and this can be incorporated in the future.

A third issue is that the current data has arbitrary units whose scale is unknown, while `HPSim` uses definite units, like amperes. To deal with this, both signals were normalized by the sum of the areas under the curve. This normalization is less sensitive to outliers and noise than scaling to a non-robust single value, like the maximum.

Finally, the data does not fully cover the entire $-180°$ to $180°$ phase range, with portions of the tails missing. To address this, synthetic data was generated to fill the missing range. This is justified on the basis that, by design of the accelerator, the beam should be a pulsed signal with approximately constant tails. Synthetic data points were generated by sampling from a normal distribution, with the mean and standard deviation estimated from the available tail data (see Fig. 2). This process was performed separately for both the PB 'on' and PB 'off' signals.

HPSim Training Data 10,000 samples with both PB 'on' and 'off'were generated. The ranges chosen can be seen in Table 1, with some variables way outside what would exactly ever be experienced in practice.

For each sample, 70 variables were derived from the two current signals, which consisted of: 1) within-series variables 2) between-series variables. The within-series variables included the mean, full width at half maximum (FWHM), the ratio of maximum to minimum values, etc. The between variables had things like cosine similarity and average distance between the two signals.

MODELS

For predicting most variables, XGBoost and Random Forest using derived features were quick to train and very accurate. However, they struggled with PBPhs and thus for that variable a transformer model was used.

XGBoost For gradient boosted decisions trees, we used the library `XGBoost` [3]. The algorithm for fitting uses the second order Newton-Raphson method, which is possible due to relative simplicity of decisions trees. The simplicity of the trees and the second order optimization makes training go far quicker than the first order gradient descent methods typically used with complex deep learning models.

Both the gradient boosted decision trees and Random Forest work on the principle that an ensemble of weak learners can be a strong learner. The idea is similar to the observation that a group of experts can often outperform any single expert.

Gradient boosting works by starting with a single decision tree. Another tree is then trained to try to anticipate the error of the first one, and correct for it. Iteratively, more trees are added.

Uncertainties here were estimated using cross validation.

Random Forest Random forest [4] builds an ensemble out of decision trees slightly differently. To create diversity, each decision tree is trained on a subsample of the data, here 80%. In addition, we choose it so that each tree in the Random Forest only looked at a fraction of a feature (80%). This helps prevent any single feature from dominating.

For both the Random Forest and the Gaussian processes, we used the software package `scikit-learn` [5].

Random forest lends itself easily to uncertainty estimate. Here they were obtained by taking the standard deviation of the predictions of the decision trees of the Random Forest.

Transformer Transformers [6] underpin many large language models, like ChatGPT. They are good for teasing out complex relations in time series data. For PBPhs, the two other models struggled. This may be due to the PB being physically furthest from the collector/absorber, combined with the wide phase range, which increases sensitivity to modeling errors. Hence, we tried a transformer model, which takes much longer to train, has less interpretability, but performed better on PBPhs. Testing on other variables, we saw no improvement in performance.

For this task, we used an encoder-type transformer, with 3 attention heads and 3 blocks of attention followed by feed-forward networks. Dropout and L2 decay were used for regularization. `Tensorflow` [7] was used to create the transformer.

RESULTS

For the XGBoost and Random Forest models, feature importance can easily be obtained by looking at how much a feature at a branch improves performance. For MBAmp, MBPhs and T1Amp, the models found it easier to use derived features coming from the PB 'off' signal, which makes sense since this simplifies the extraction. In addition, for PBAmp, the models found the between-series variables the most important features. Again, this makes sense given the difference between the two beams is coming from the PB.

The predictions from the models given the phase scan data can be seen in Table 2. For PBPhs, the transformer model's prediction was used. For the others, the predictions of the XGBoost and Random Forest were averaged. The results can be seen in Fig. 3. Both the PB 'on' and 'off' match qualitatively, though the 'on' data predictions shows a wider peak and a narrow plateau.

One important test we performed was using the absorber/collector data varying the PB phase. This data is very noisy and was therefore excluded from training. However, it can serve as an independent test of the parameters obtained. As can be seen in Fig. 4, the predicted curve based

Table 2: Predictions and Uncertainties From the Models

	XGBoost	Random Forest	Transformer
MBAmp	11.654 ± 0.037	11.950 ± 0.043	–
MBPhs	87 ° ± 10 °	88.1 ° ± 1.5 °	–
PBAmp	4.32 ± 0.12	4.66 ± 0.22	–
T1Amp	25.385 ± 0.008	25.376 ± 0.0013	–
PBPhs	–	–	-134.8 ° ± 6.9 °

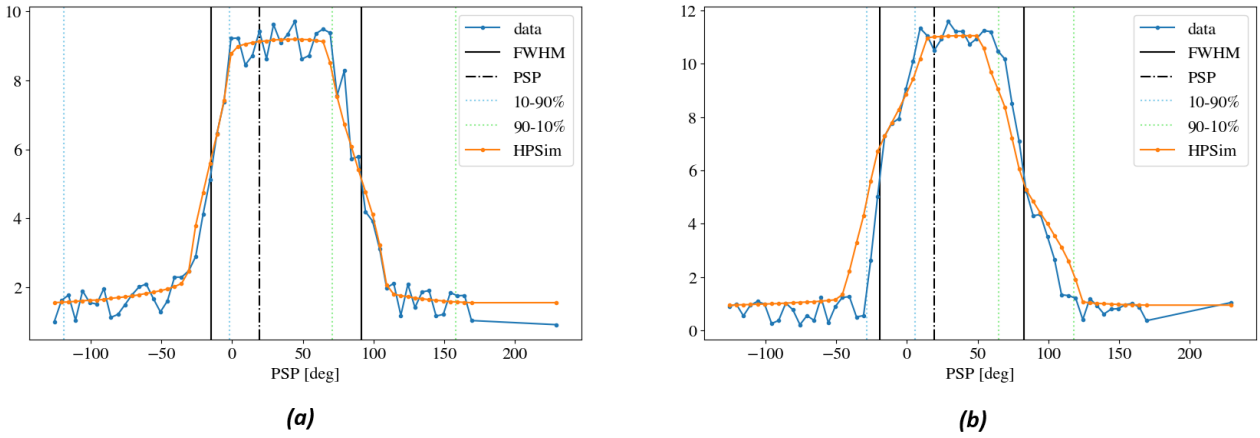

Figure 3: Phase scan data and HPSim run with predicted variables. (a) PB 'off' (b) PB 'on'.

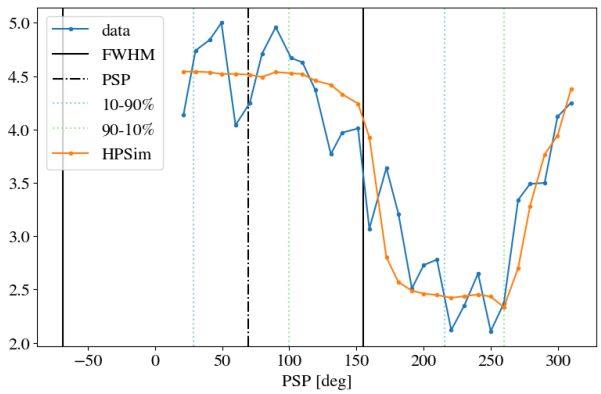

Figure 4: Current vs phase data from the PB absorber/collector and the generated curve from HPSim using the predicted variables. This data was not used at all during the training process, thus serves a good independent test for model predictions.

on the predicted parameters matches well, despite the model not being trained whatsoever on this data.

CONCLUSION

We showed that ML can speed the process of calibrating the front-end of LANSCE while requiring minimal human input. Much of the process has been coded to be automated and can be used in the future. With further optimization, we expect automation to reduce what previously required a month of expert labor to 2–3 days of primarily computational work.

The uncertainties presented here reflect model uncertainties. To incorporate data uncertainties, the Gaussian process fit can be used to sample from a distribution of functions, rather than just using the mean functions. Improvements can be made by iteratively improving upon the models' prediction in HPSim, perhaps with Bayesian optimization. The models tend to rely more heavily on features derived from the PB 'off' signal. As a result, the PB 'on' signal is modeled less accurately, as seen in Fig. 3. Attempts to improve the PB 'on' fit can be made by using models that only use the 'on' data or attempt to force the models to incorporate features from this signal.

REFERENCES

[1] X. Pang and L. Rybarcyk, "GPU accelerated online multi-particle beam dynamics simulator for ion linear particle accelerators", *Comput. Phys. Commun.*, vol. 185, no. 3, pp. 744–753, 2014. doi:10.1016/j.cpc.2013.10.033

[2] E. C. Huang, P. M. Anisimov, A. J. Braido, M. J. Kay, L. J. Rybarcyk, and J. K. K. Quemuel, "Online multi-particle model for LANSCE physics tune-up with HPSim", in *Proc. LINAC'24*, Chicago, IL, USA, Aug. 2024, pp. 231–234. doi:10.18429/JACoW-LINAC2024-MOPB084

[3] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system", in *Proc. KDD'16*, San Francisco, California, USA, Aug. 2016, pp. 785–794. doi:10.1145/2939672.2939785

[4] L. Breiman, "Random forests", *Mach. Learn.*, vol. 45, pp. 5–32, 2001. doi:10.1023/A:1010933404324

[5] F. Pedregosa *et al.*, "Scikit-learn: machine learning in Python", *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[6] A. Vaswani *et al.*, “Attention is all you need”, in *Proc. NeurIPS’17*, Long Beach, CA, USA, Dec. 2017.

[7] M. Abadi *et al.*, “Tensorflow: large-scale machine learning on heterogeneous distributed systems”, *arXiv*, 2016. doi:10.48550/arXiv.1603.04467