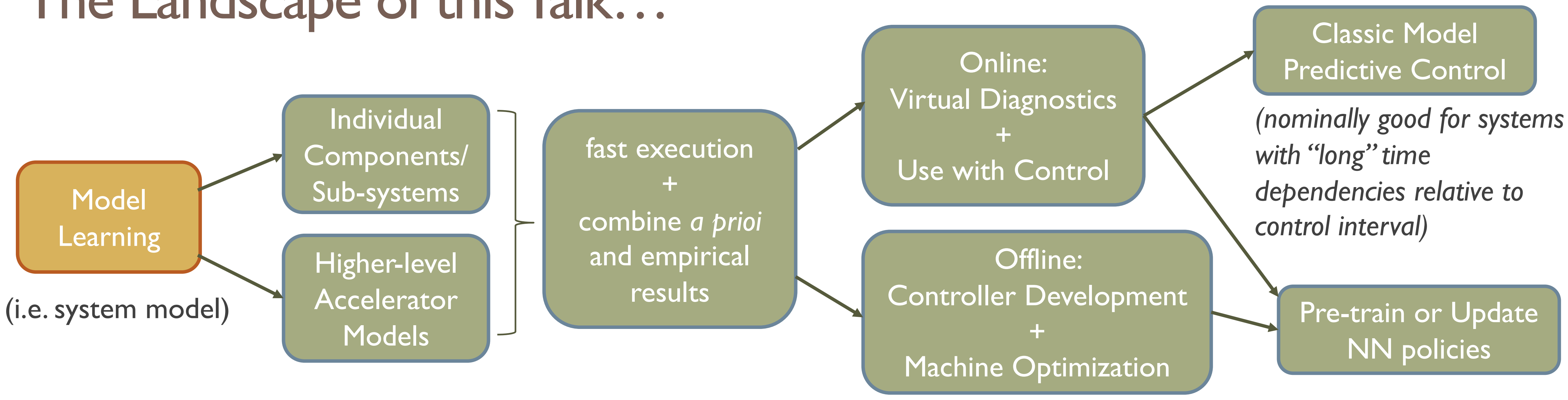
Experience with Model Predictive Control and Model-Based Reinforcement Learning

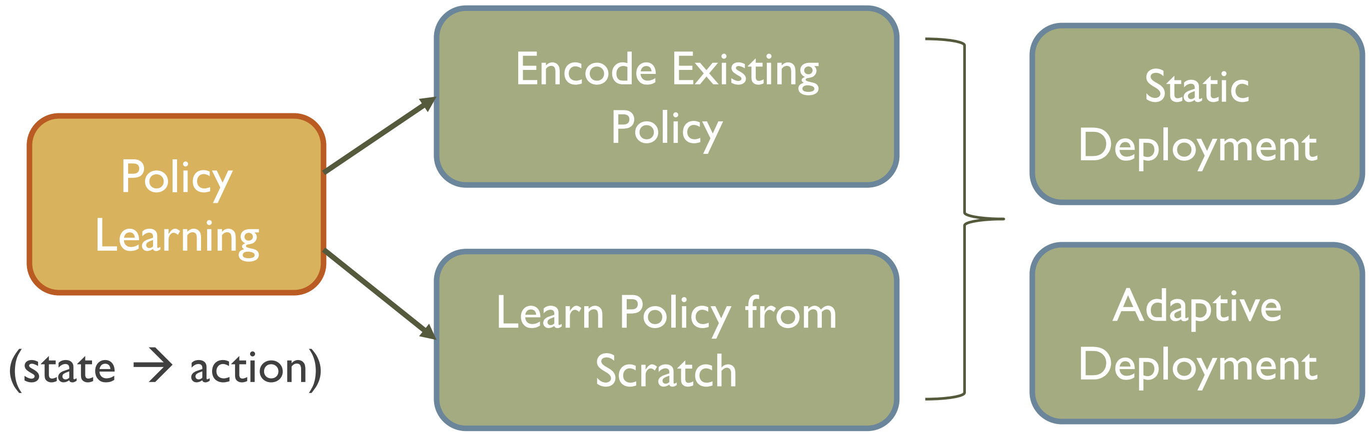
Auralee Edelen

Mar. 1 2018, ICFA Workshop on ML for Particle Accelerators

Work with Sandra Biedron, Daniel Bowring, Brian Chase, David Douglas, Jonathan Edelen, Chip Edstrom, Denise Finstrom, Henry Freund, Stephen Milton, Dennis Nicklaus, Jinhao Ruan, Jim Steimel, Chris Tennant, Peter van der Slot, and many others

The Landscape of this Talk…





- *If adaptive ML policy for tuning: gain some of the same advantages as using direct online optimization + remember previous solutions / interpolate (useful if drift is small?)*

Online Modeling

- Use a machine model during operation

- Ideally:
  - Fast-executing, but accurate enough to be useful
  - Use measured inputs directly from machine
  - Combine *a priori* knowledge + learned parameters

- Applications:
  - A tool for operators + virtual diagnostic
  - Predictive control
  - Help flag aberrant behavior
  - *Bonus: control system development*

Online Modeling

- Use a machine model during operation

- Ideally:
  - Fast-executing, but accurate enough to be useful
  - Use measured inputs directly from machine
  - Combine *a priori* knowledge + learned parameters

- Applications:
  - A tool for operators + virtual diagnostic
  - Predictive control
  - Help flag aberrant behavior
  - *Bonus: control system development*

One approach: faster modeling codes

Simpler models (tradeoff with accuracy)

analytic calculations *e. g. J. Galambos, et al., HPPA5, 2007*

Parallelization and GPU-acceleration of existing codes

PARMILA → HPSim *X. Pang, PAC13, MOPMA13*

elegant *I.V. Pogorelov, et al., IPAC15, MOPMA035*

Improvements in underlying modeling algorithms

Online Modeling

- Use a machine model during operation
- Ideally:
  - Fast-executing, but accurate enough to be useful
  - Use measured inputs directly from machine
  - Combine *a priori* knowledge + learned parameters
- Applications:
  - A tool for operators + virtual diagnostic
  - Predictive control
  - Help flag aberrant behavior
  - *Bonus: control system development*

One approach: faster modeling codes

Simpler models (tradeoff with accuracy)

analytic calculations *e. g. J. Galambos, et al., HPPA5, 2007*

Parallelization and GPU-acceleration of existing codes

PARMILA → HPSim *X. Pang, PAC13, MOPMA13*

elegant *I.V. Pogorelov, et al., IPAC15, MOPMA035*

Improvements in underlying modeling algorithms

Another approach: machine learning model

Once trained, neural networks can execute quickly

Train on results from slow, high-fidelity simulations

Train on measured results

Online Modeling

- Use a machine model during operation

- Ideally:
  - Fast-executing, but accurate enough to be useful
  - Use measured inputs directly from machine
  - Combine *a priori* knowledge + learned parameters

- Applications:
  - A tool for operators + virtual diagnostic
  - Predictive control
  - Help flag aberrant behavior
  - *Bonus: control system development*

One approach: faster modeling codes

Simpler models (tradeoff with accuracy)

analytic calculations — *e. g. J. Galambos, et al., HPPA5, 2007*

Parallelization and GPU-acceleration of existing codes

PARMILA → HPSim — *X. Pang, PAC13, MOPMA13*

elegant — *I.V. Pogorelov, et al., IPAC15, MOPMA035*

Improvements in underlying modeling algorithms

(fractions of a second)

Another approach: machine learning model

Once trained, neural networks can execute quickly

Train on results from slow, high-fidelity simulations

Train on measured results

Yields a fast-executing model that can be used operationally, but approximates behavior from slower, high-fidelity simulations (e.g. PIC codes, plasma acc., space charge)

Online Modeling

- Use a machine model during operation
- Ideally:
  - Fast-executing, but accurate enough to be useful
  - Use measured inputs directly from machine
  - Combine *a priori* knowledge + learned parameters
- Applications:
  - A tool for operators + virtual diagnostic
  - Predictive control
  - Help flag aberrant behavior
  - *Bonus: control system development*

One approach: faster modeling codes

Simpler models (tradeoff with accuracy)

analytic calculations — *e. g. J. Galambos, et al., HPPA5, 2007*

Parallelization and GPU-acceleration of existing codes

PARMILA → HPSim — *X. Pang, PAC13, MOPMA13*

elegant — *I.V. Pogorelov, et al., IPAC15, MOPMA035*

Improvements in underlying modeling algorithms

Another approach: machine learning model

(fractions of a second)

Once trained, neural networks can execute quickly

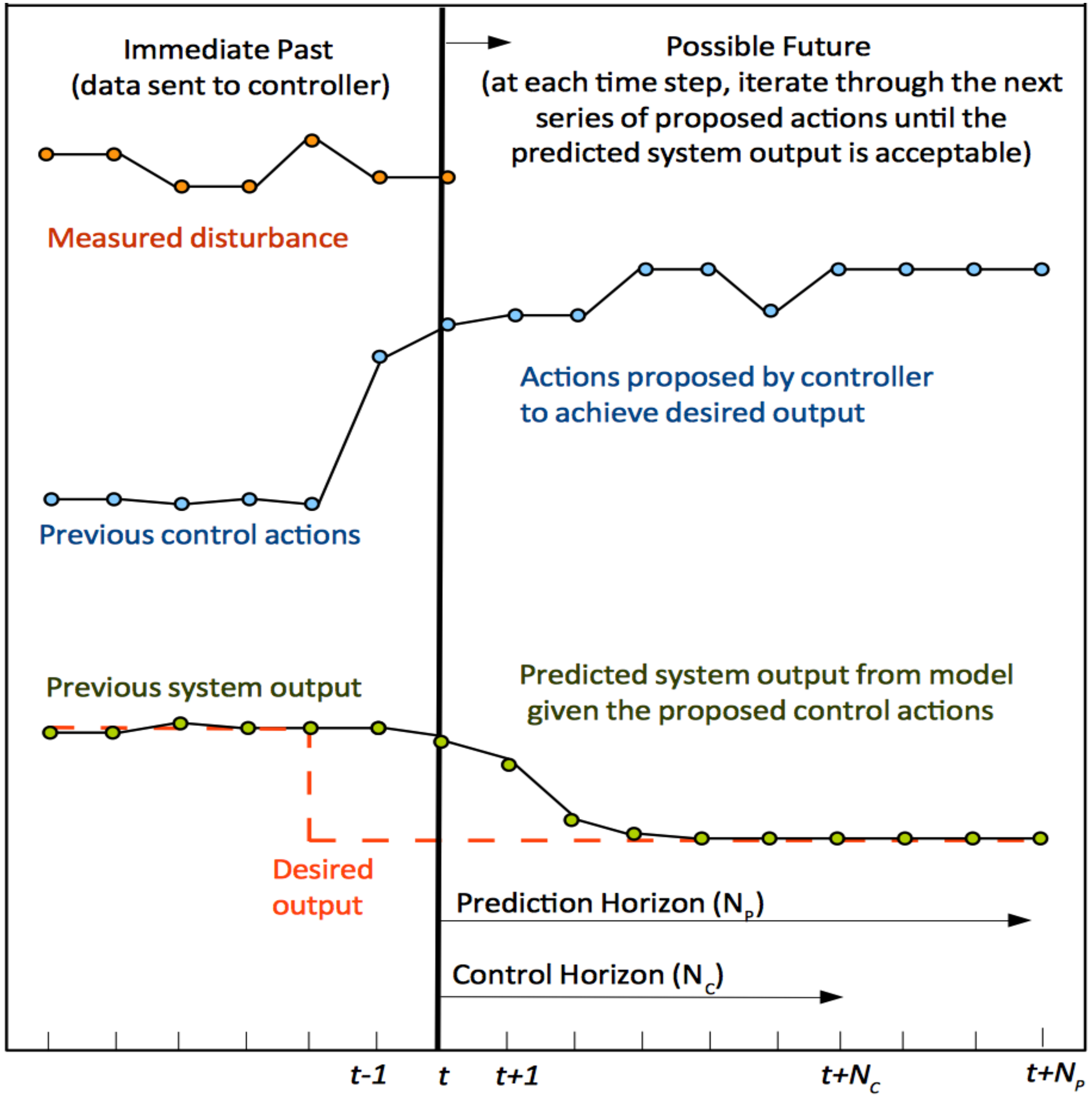
Train on results from slow, high-fidelity simulations

Train on measured results

Yields a fast-executing model that can be used operationally, but approximates behavior from slower, high-fidelity simulations (e.g. PIC codes, plasma acc., space charge)

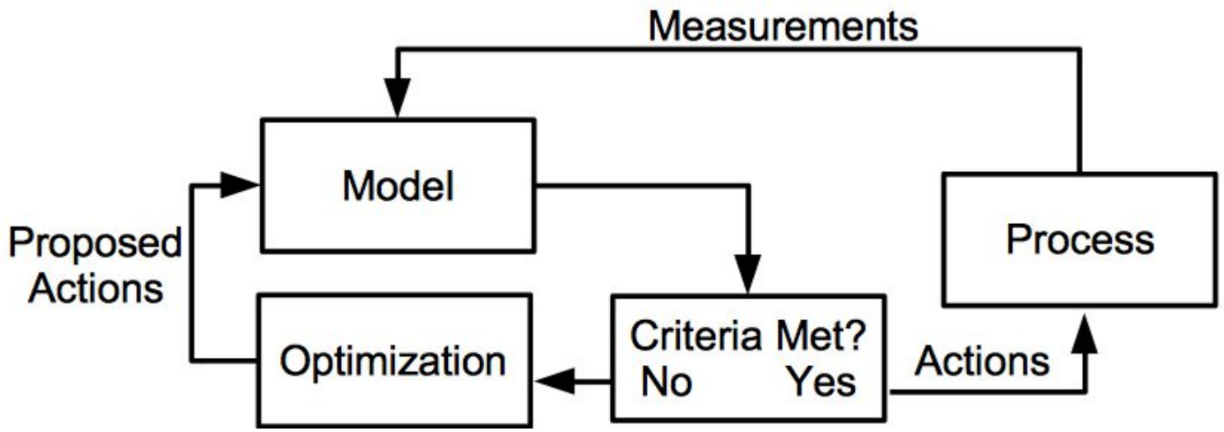
An initial study at Fermilab:
A. L. Edelen, et al. NAPAC16, TUPOA51
One PARMELA run with 2-D space charge: ~ 20 minutes
Neural network model: ~ a millisecond

Model Predictive Control (Prediction + Planning)

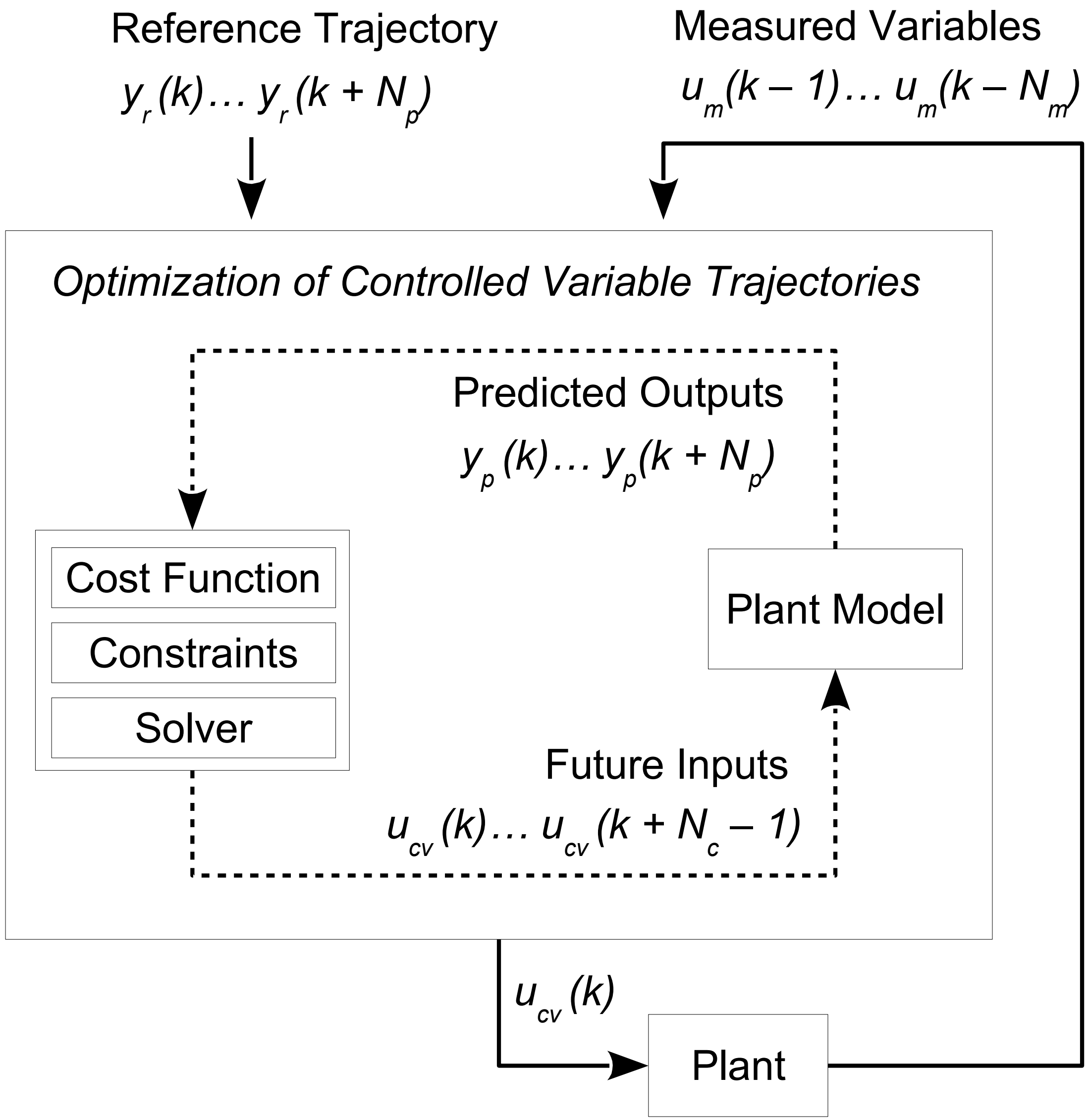


Basic concept:

1. Use a predictive model to assess the outcome of possible future actions
2. Choose the best series of actions
3. Execute the first action
4. Gather next time step of data
5. Repeat



Model Predictive Control (Prediction + Planning)



N_m previous measurements

N_p future time steps predicted

N_c future time steps controlled

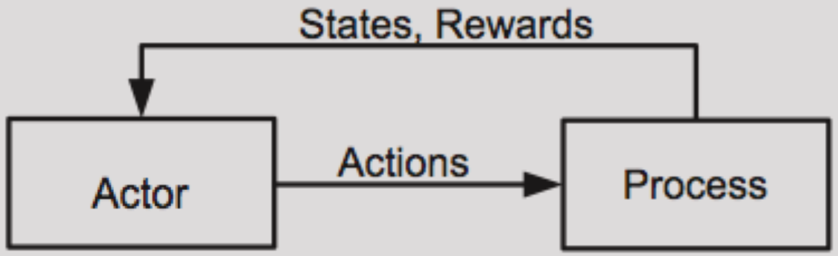
$$\sum_{i=1}^{N_p} \{w_y [y_r(k+i) - y_p(k+i)]\}^2$$
(output variable targets)

$$\sum_{j=1}^{ncv} \sum_{i=0}^{N_p-1} \{w_{u,j} [u_j(k+i) - u_{j,ref}(k+i)]\}^2$$
(controllable variable targets)

$$\sum_{j=1}^{ncv} \sum_{i=0}^{N_p-1} \{w_{\Delta u,j} [u_j(k+i) - u_j(k+i-1)]\}^2$$
(movement size)

Neural Network Policies and Reinforcement Learning

Actor-only Methods

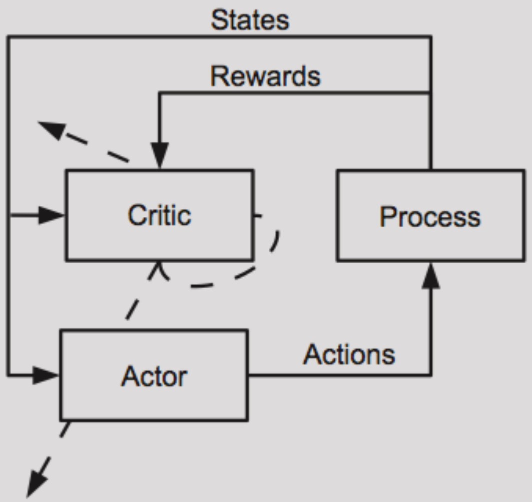


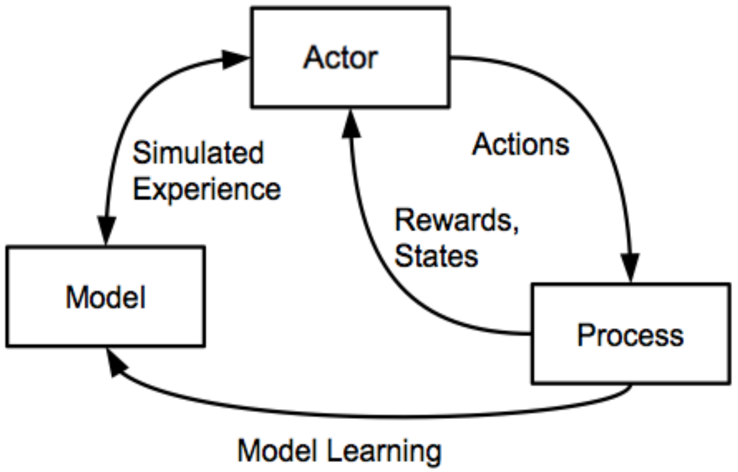
- Actor is a control policy
- Maps states to actions
- Reward provides training signal

Actor-Critic Methods

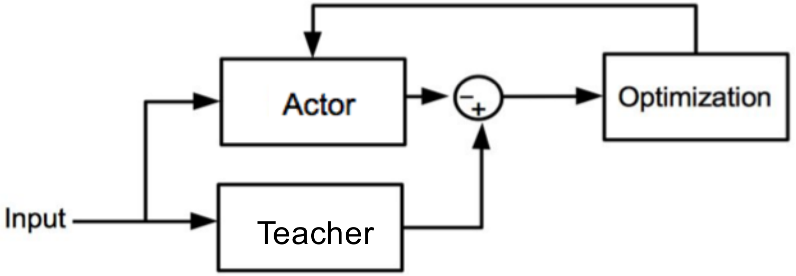
- Critic maps states or state/action pairs to an estimate of long-term reward
- Could be a NN, tabular, etc.
- Critic provides training signal to actor

Without actor: use an optimization algorithm with the critic





Can train on models first to get a good initial solution before deployment



Can use supervised learning to first approximate the behavior of a different control policy

A few examples …

Dealing with "Long-Term" Time Dependencies: Resonant Frequency Control in Normal Conducting Cavities

RF electron gun at the Fermilab Accelerator Science and Technology (FAST) facility

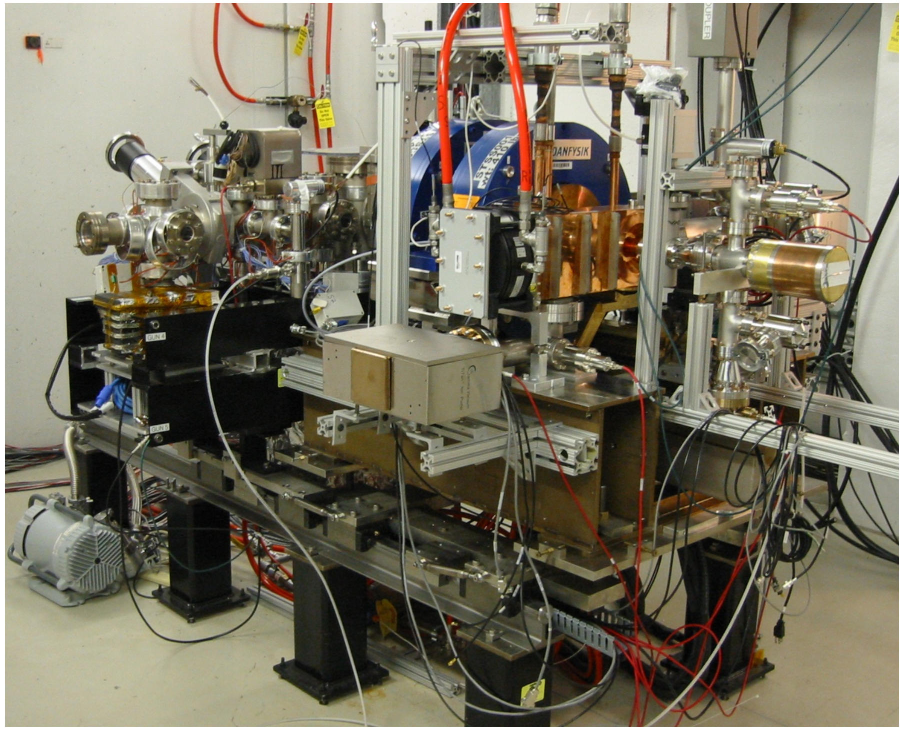


Photo: P. Stabile

Radio frequency quadrupole (RFQ) for the PIP-II Injector Test

Photo: J. Steimel

"long term" in this case means responses lasting many minutes (e.g. 30), with control actions at 0.5 Hz and 1 Hz

Why does this matter for normal-conducting cavities?

The LLRF system will compensate for detuning by increasing forward power

Why does this matter for normal-conducting cavities?

The LLRF system will compensate for detuning by increasing forward power

But…

- Ability to do this bounded by the amplifier specs
- If detuned beyond RF overhead → *interrupt normal operations*
- RF overhead adds to initial machine cost and footprint
- Using additional RF power → *increasing operational cost*
- Increased waste heat into cooling system → *increasing operational cost*

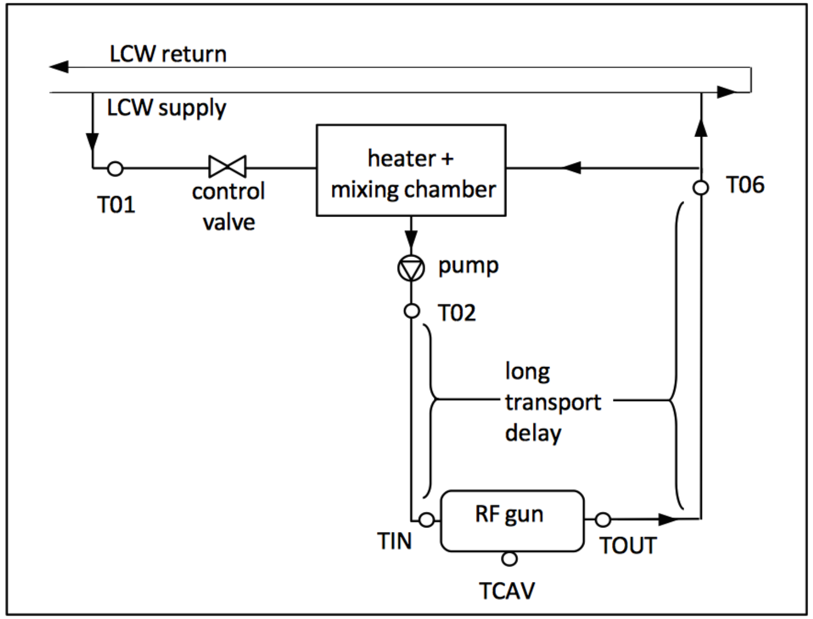
Temperature Control for the RF Photoinjector at FAST

Resonant frequency controlled via temperature

PID control is undesirable in this case:

- Long transport delays and thermal responses
- Recirculation leads to secondary impact of disturbances
- Two controllable variables: heater power + valve aperture

Gun Water System Layout



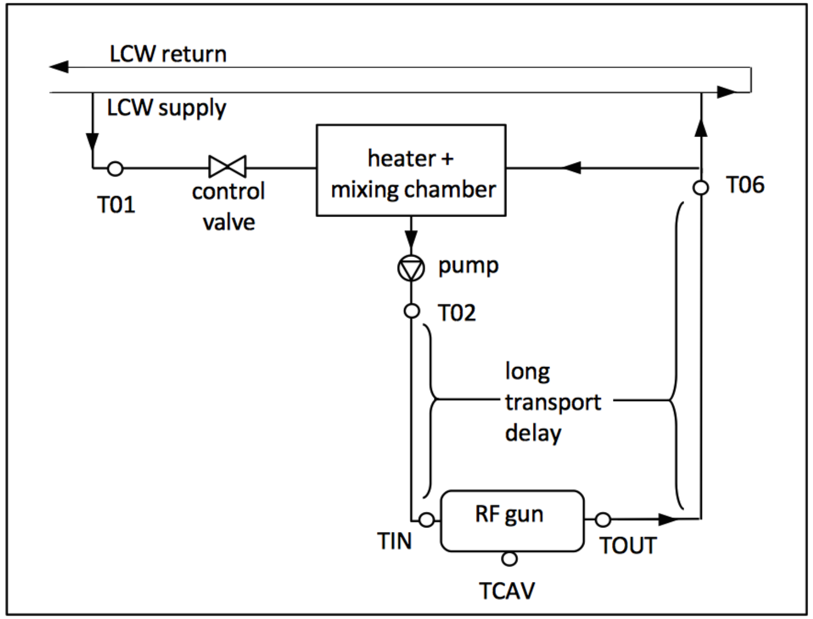
Temperature Control for the RF Photoinjector at FAST

Resonant frequency controlled via temperature

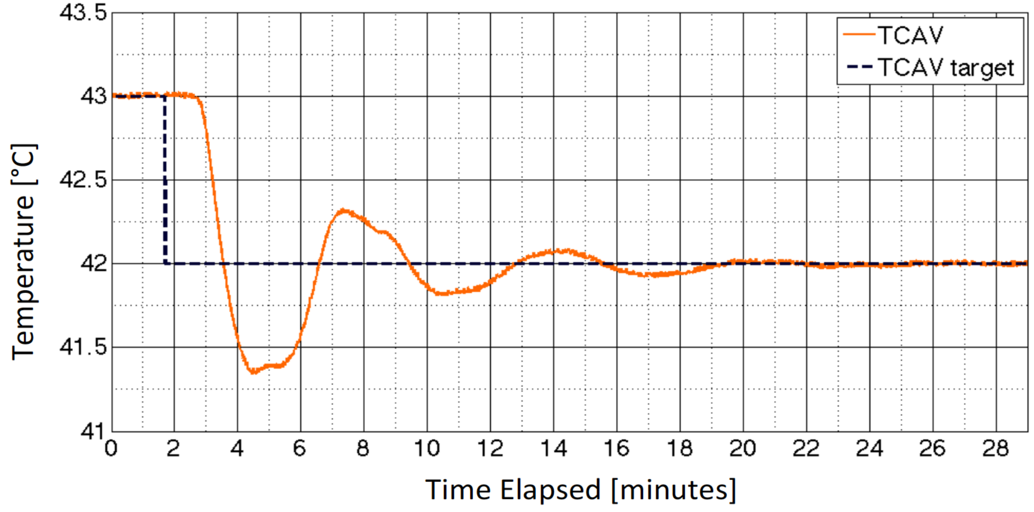
PID control is undesirable in this case:
- Long transport delays and thermal responses
- Recirculation leads to secondary impact of disturbances
- Two controllable variables: heater power + valve aperture

Applied model predictive control (MPC) with a neural network model trained on measured data: ~ 5x faster settling time + no large overshoot

Gun Water System Layout

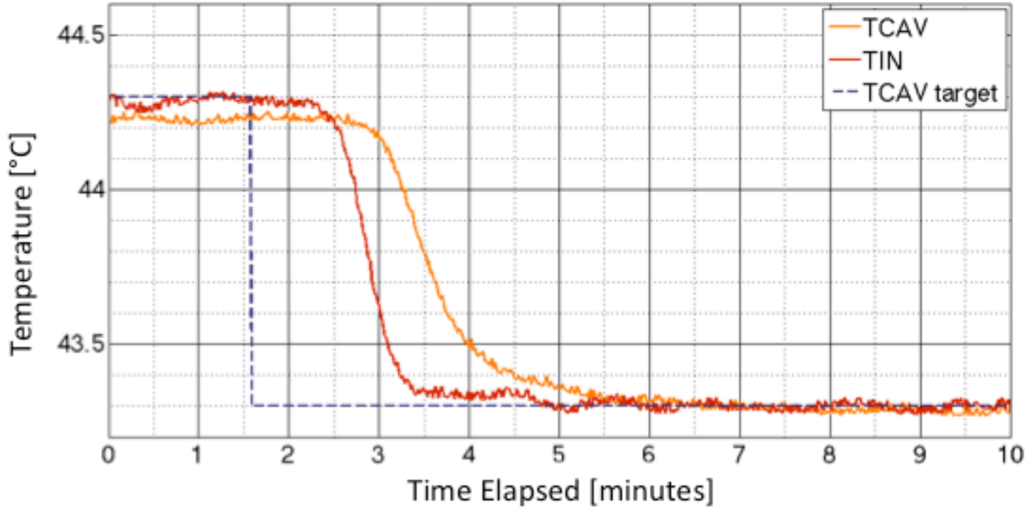


Existing Feedforward/PID Controller



Note that the oscillations are largely due to the transport delays and water recirculation, rather than PID gains

Model Predictive Controller



A. L. Edelen et al., TNS, vol. 63, no. 2, 2016 A.L. Edelen et al., IPAC '15

PIP-II Injector Test RFQ

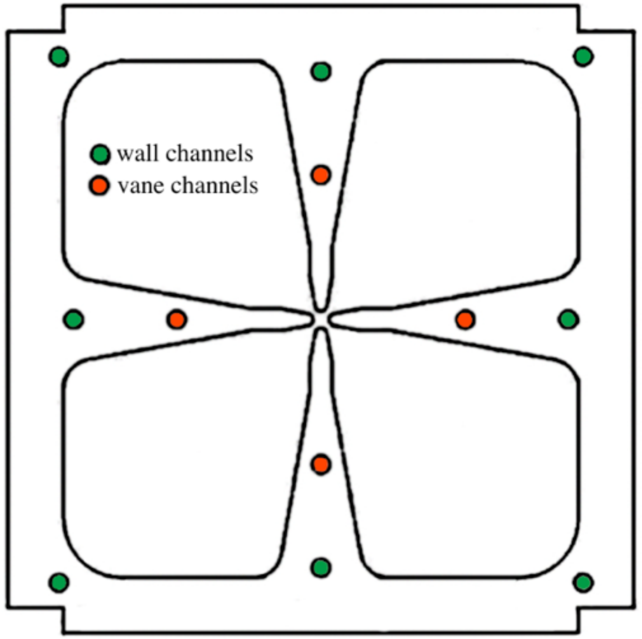
variable heating

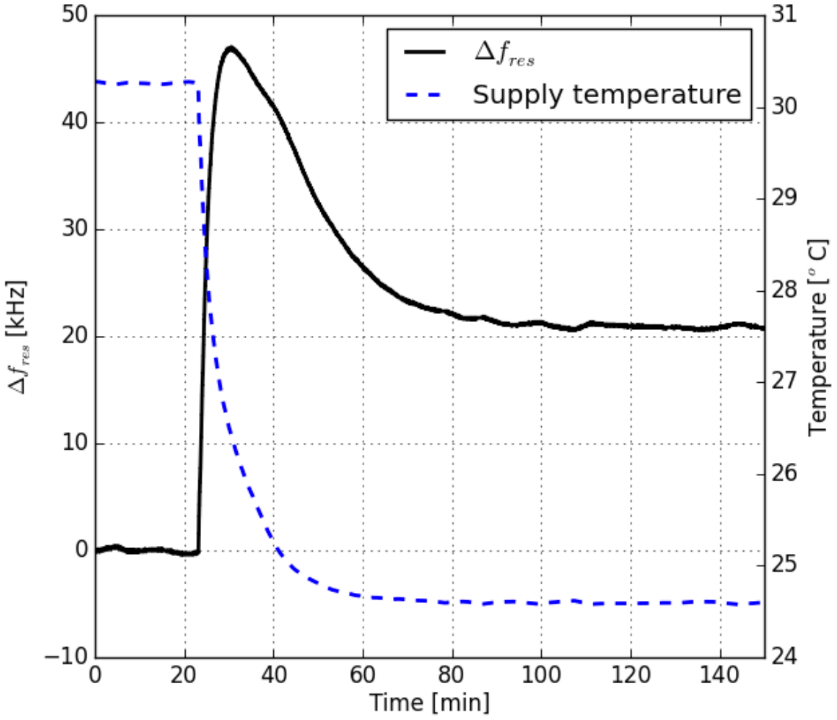
Specification for GDR: 3-kHz maximum frequency shift

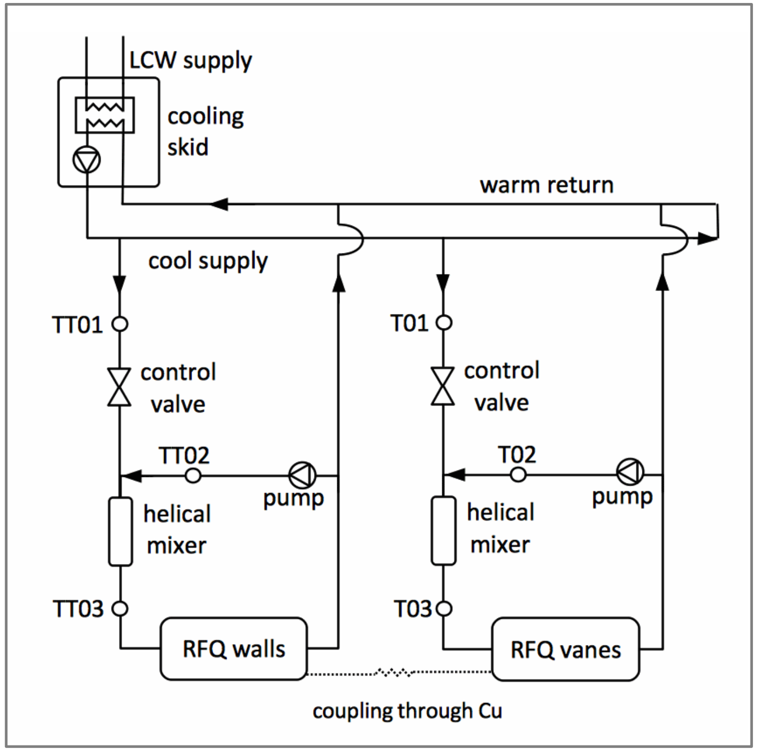
Range of RF duty factors and pulse patterns (up to CW)

-16.7 kHz/°C in the vanes and 13.9 kHz/°C in the walls*

* *A. R. Lambert et al., IPAC'15, WEPTY045*

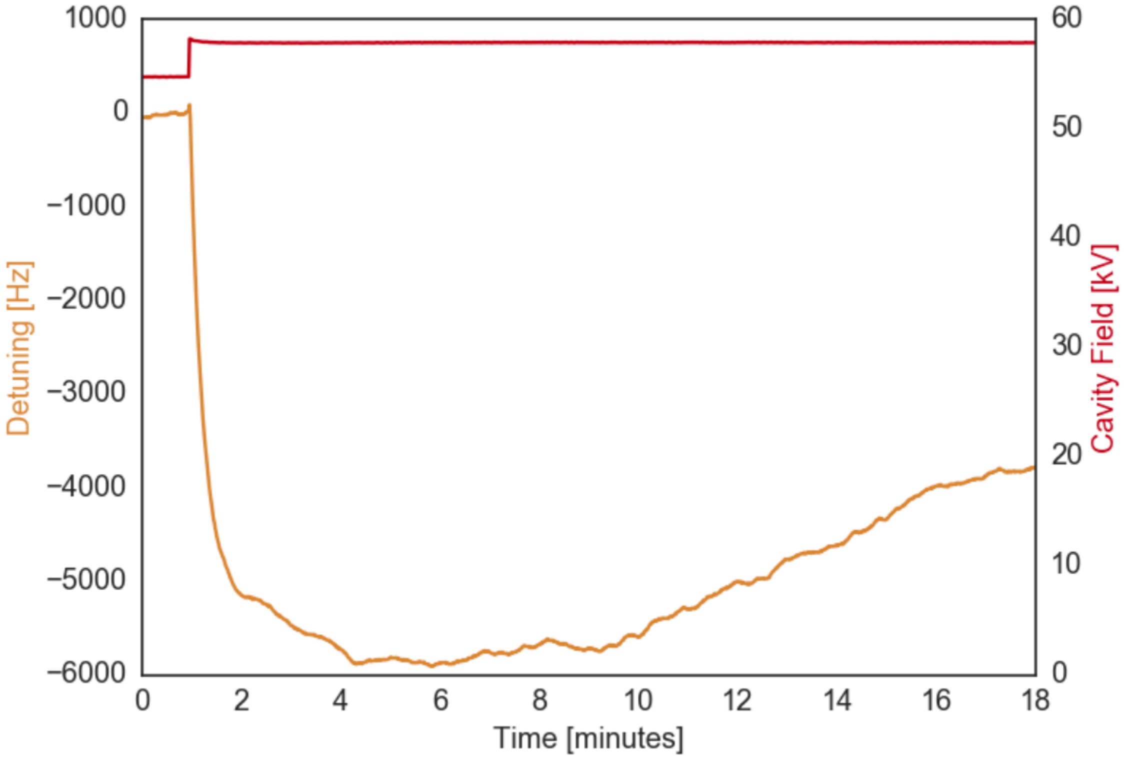




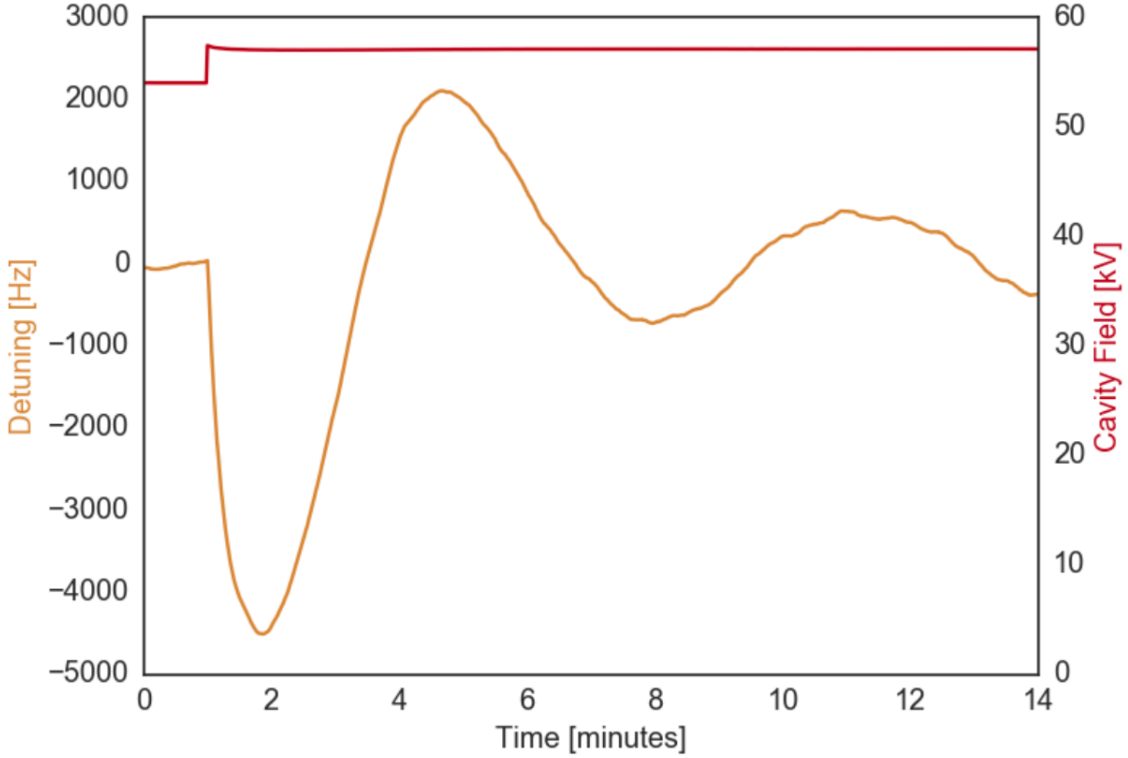


Added Motivation: RFQ Detuning in CW Mode

For a small change in cavity field (55 kV to 58 kV)…

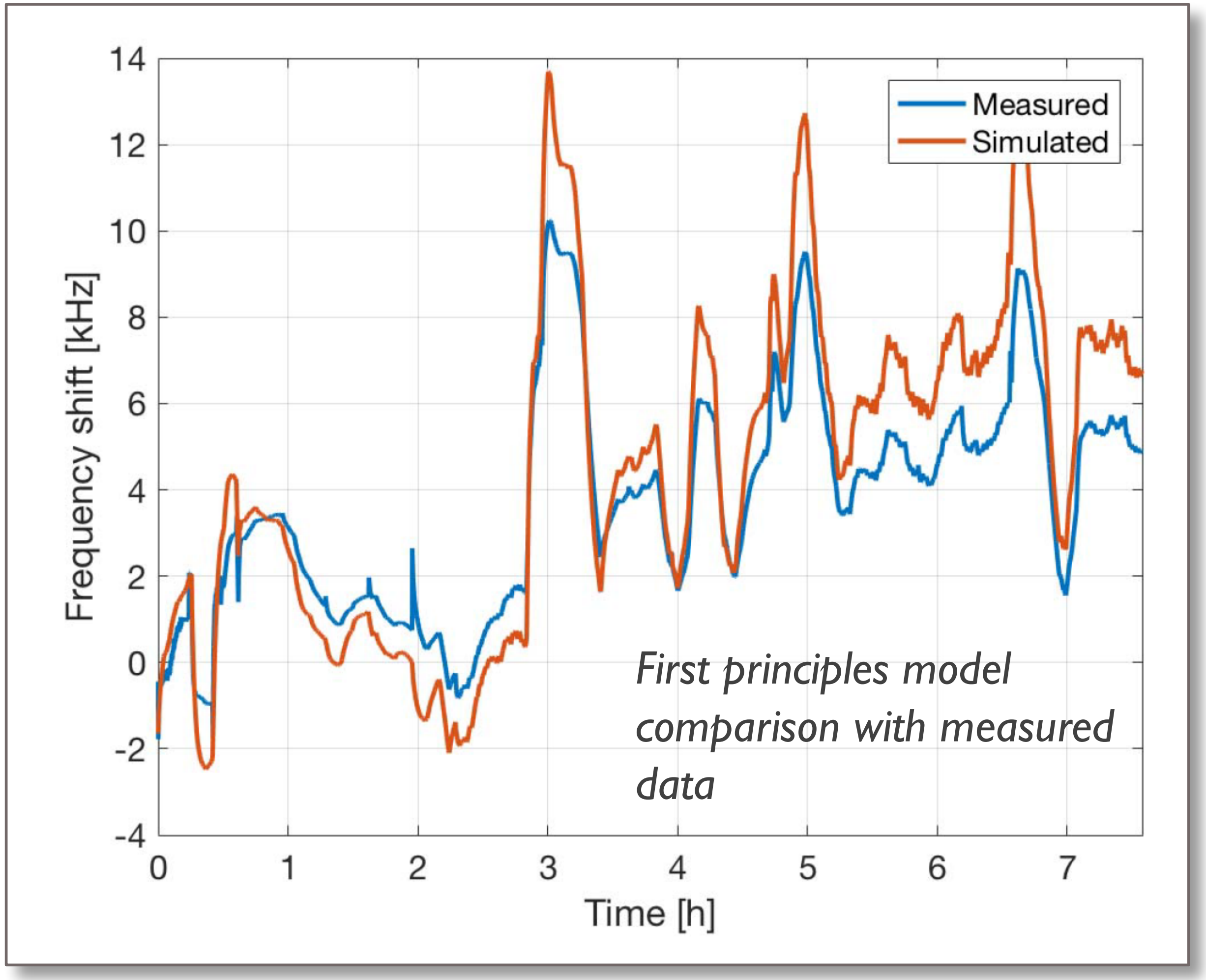


Uncontrolled



PI Frequency Control

Created a fast first-principles model, so why not use that in MPC instead of a NN?



Model needs to be sufficiently accurate for MPC

Assessed performance using measured input data:
4 ms RF pulse duration, 10 Hz rep rate
variety of valve and power settings

1.67 kHz RMS error
4.01 kHz max error
Maximum acceptable detuning is 3 kHz

Not accurate enough for control with MPC!

even after extensive tuning of uncertain parameters using an optimizer

Also looked at a linear learned model: still too poor
1.13 kHz RMS, 2.66 kHz max error

J. Edelen, A. Edelen, et al. TNS 64, vol. 2, (2017)

Initial NN Modeling for RFQ: Same as for FAST

*wanted to make sure we **could** model the response before moving forward*

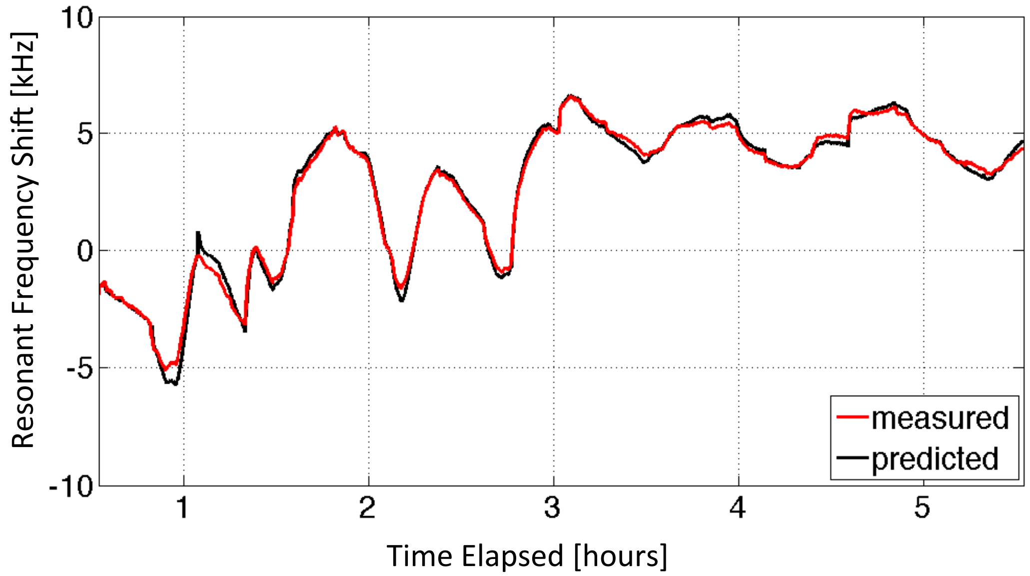
Feed-forward, Fully-connected Network

x_{t-hist} … x_t → NN → f_{t+1}

Two hidden layers: 25 and 7 nodes

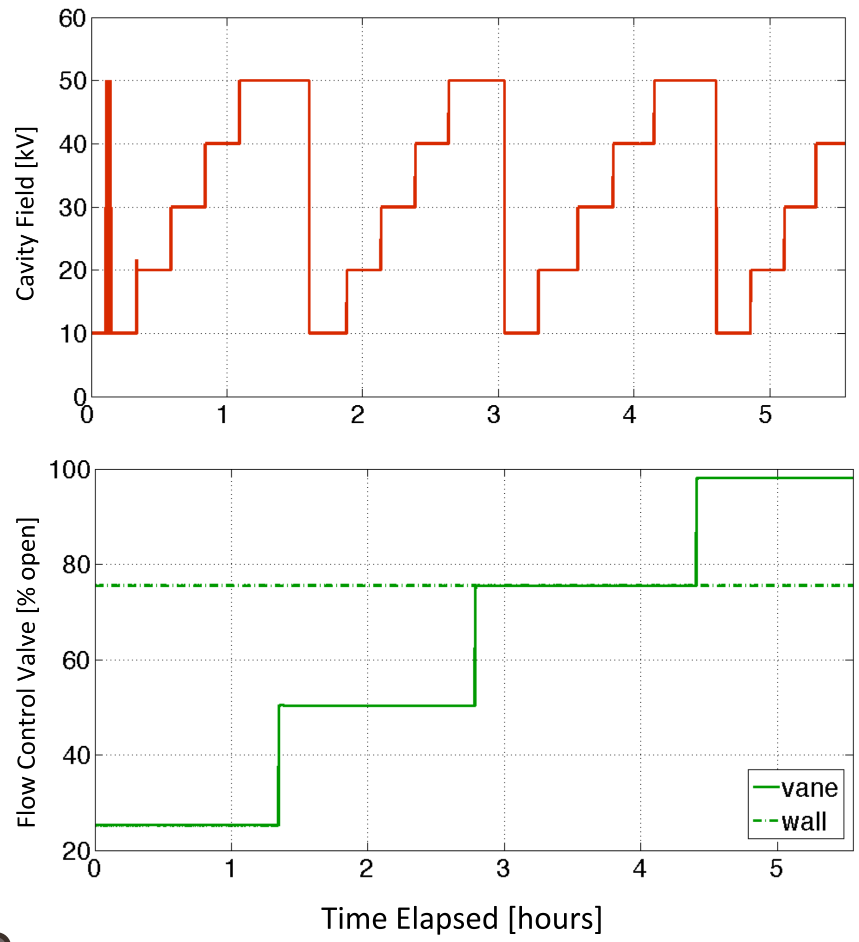
hist: 30 minutes at 1 Hz

x includes:
- Valve settings
- Average RF power
- Water temperatures
- Cave temperature
- Cave humidity



Mean Absolute Error

346 Hz – test set
98 Hz – validation set
115 Hz – across all sets



Training Data

~ 64 hours of measurements
Scanned average RF power, valves
Includes RF trips, startup/shutdown

A.L. Edelen et al., IPAC '16

Recurrent NN is physically well-motivated

Recurrent NN is physically well-motivated

4000 prior time steps x7 features
→ 600 time steps prediction horizon

Recurrent NN is physically well-motivated

Long Short-term Memory Network?

4000 prior time steps x7 features
→ 600 time steps prediction horizon

Recurrent NN is physically well-motivated

~~Long~~ Short-term Memory Network?

4000 prior time steps x7 features
→ 600 time steps prediction horizon

200-600 prior samples

Recurrent NN is physically well-motivated

~~Long~~ Short-term Memory Network?

4000 prior time steps x7 features
→ 600 time steps prediction horizon

200-600 prior samples

Can still be made to work → but not stably enough for online updating

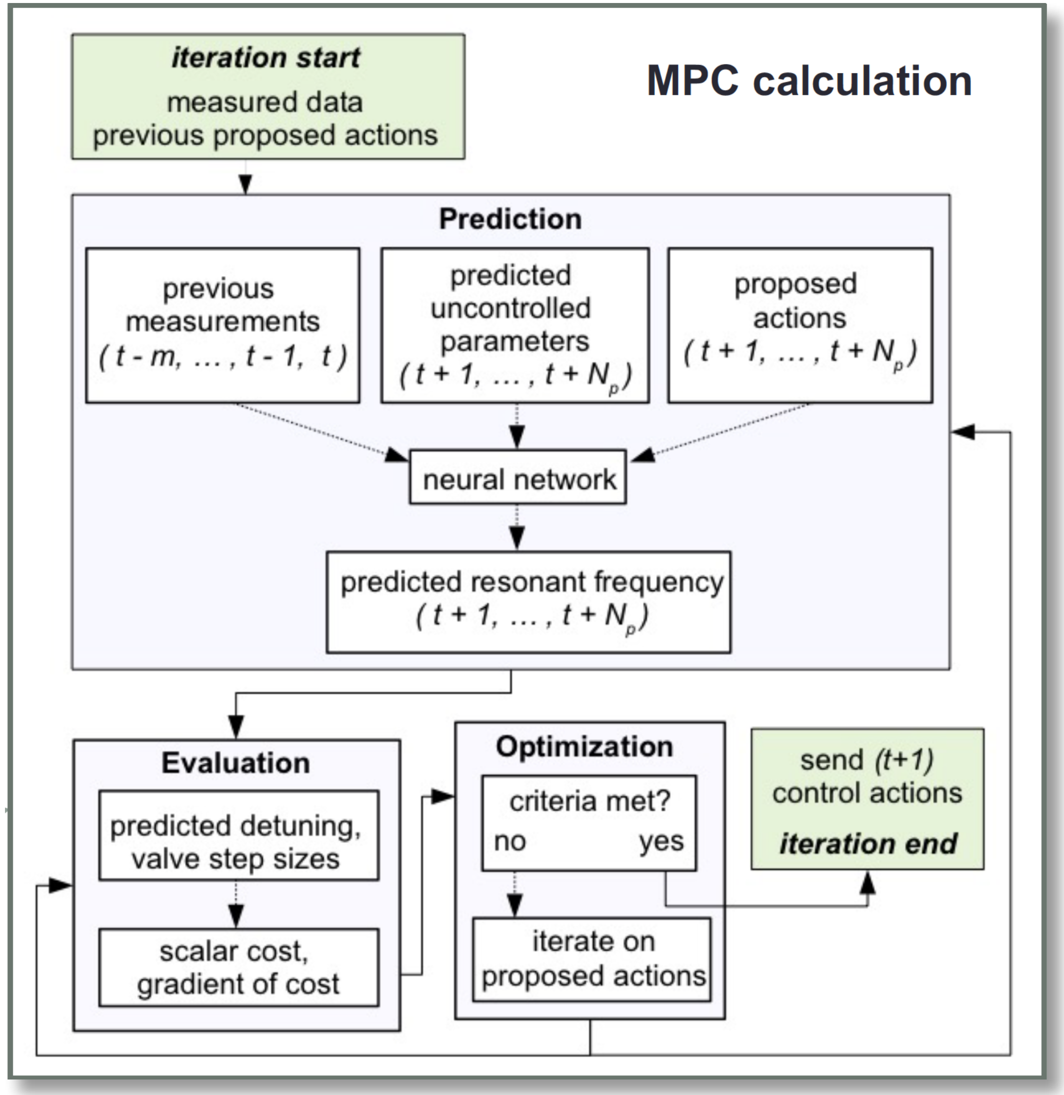
A Computationally Efficient Model for Execution?

- Had to run on Fermilab controls network machines
- This means: limited processing speed and memory
- Found that RNN is too computationally intensive
- Found that cycling over one-step-ahead predictions of a feedforward net is too computationally intensive
- Limited funds to purchase/support a new computer

A Computationally Efficient Model for Execution?

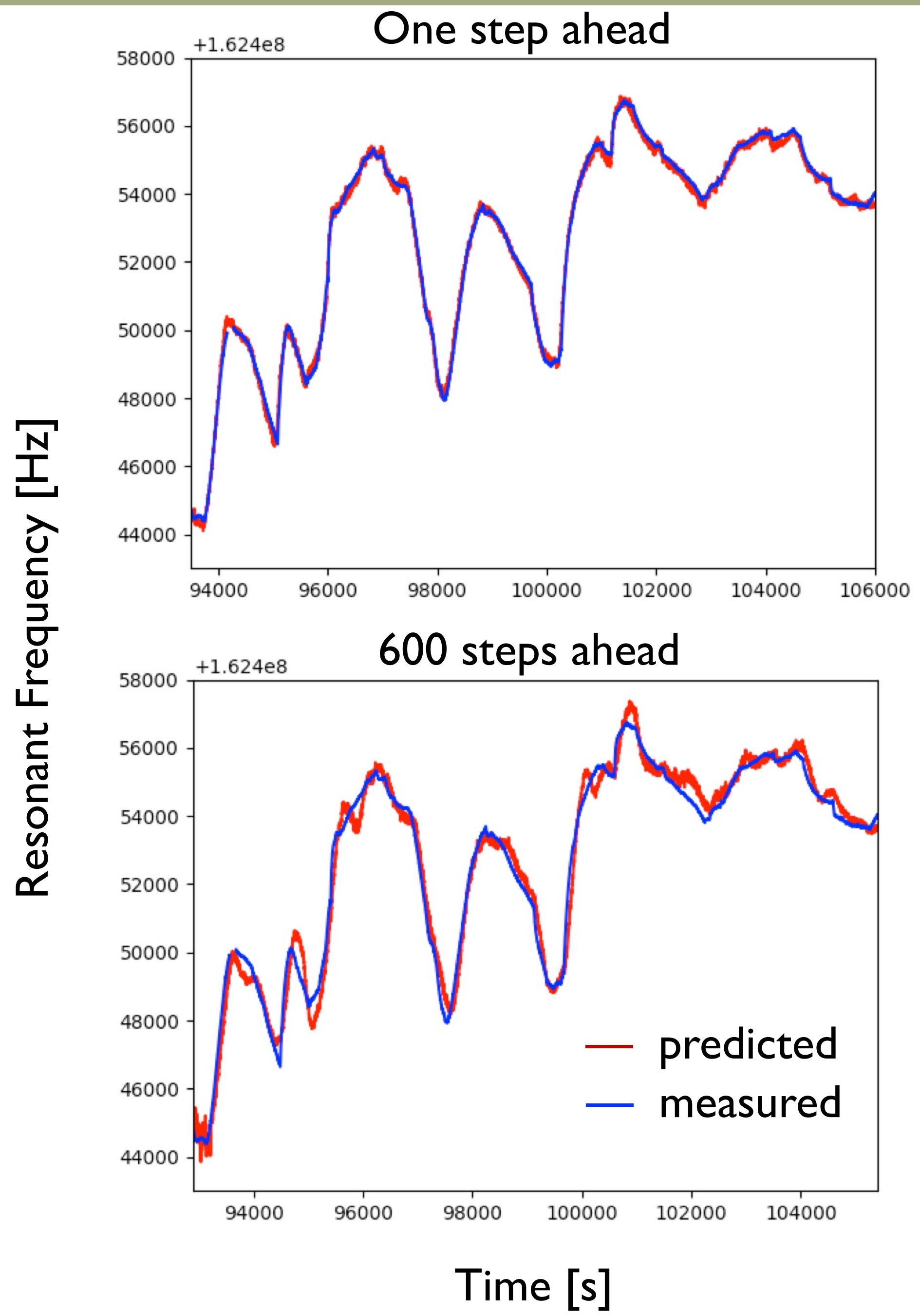
- Had to run on Fermilab controls network machines
- This means: limited processing speed and memory
- Found that RNN is too computationally intensive
- Found that cycling over one-step-ahead predictions of a feedforward net is too computationally intensive
- Limited funds to purchase/support a new computer

- First solution: sparser input history/prediction horizon
- Second solution: predict entire horizon in one iteration
- Third solution: NN policy mimicking MPC



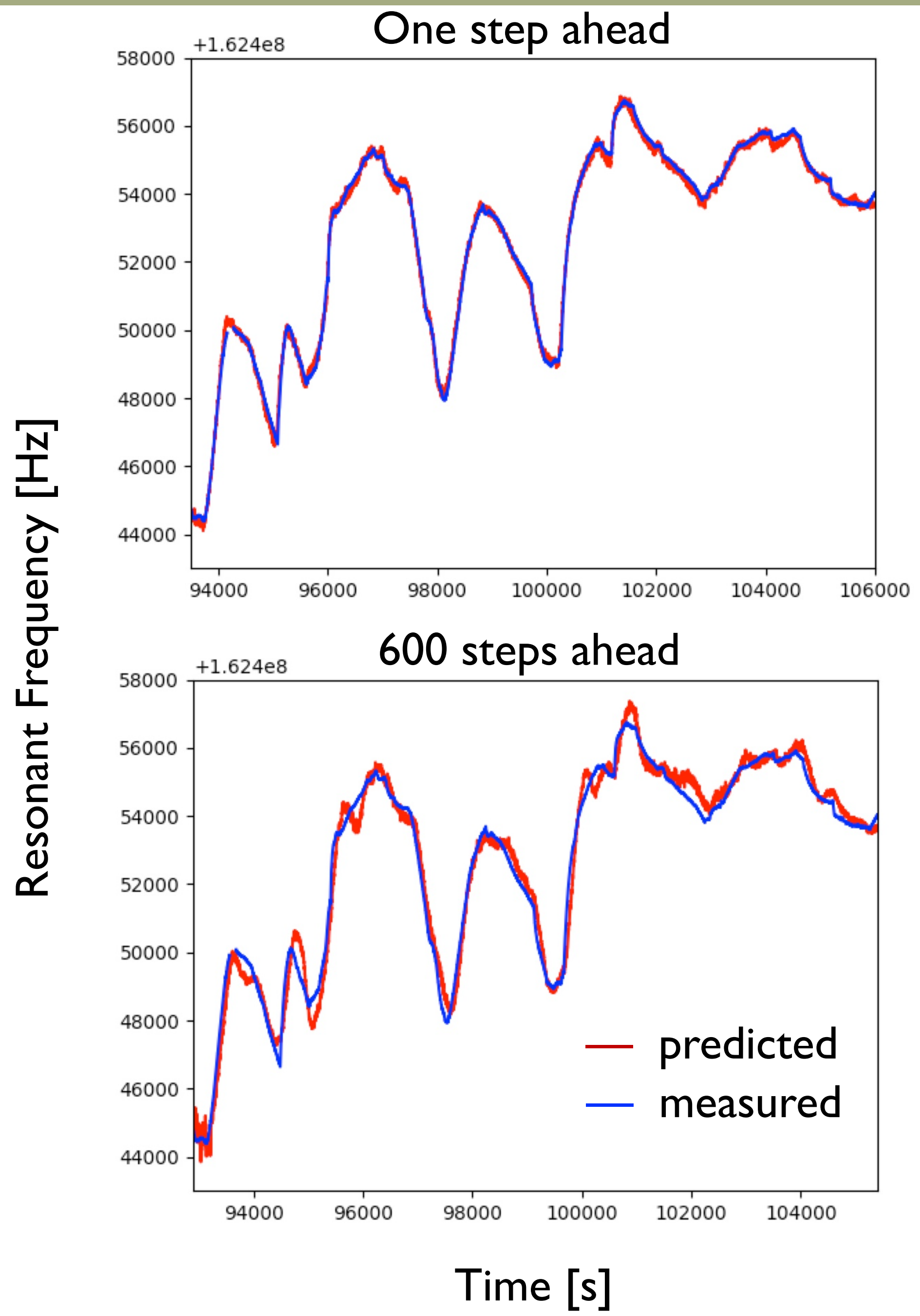
Other Stumbling Blocks…

- One-step ahead: 106 Hz MAE, 796 Hz max
- 600-step ahead: 339 Hz MAE, 1588 Hz max



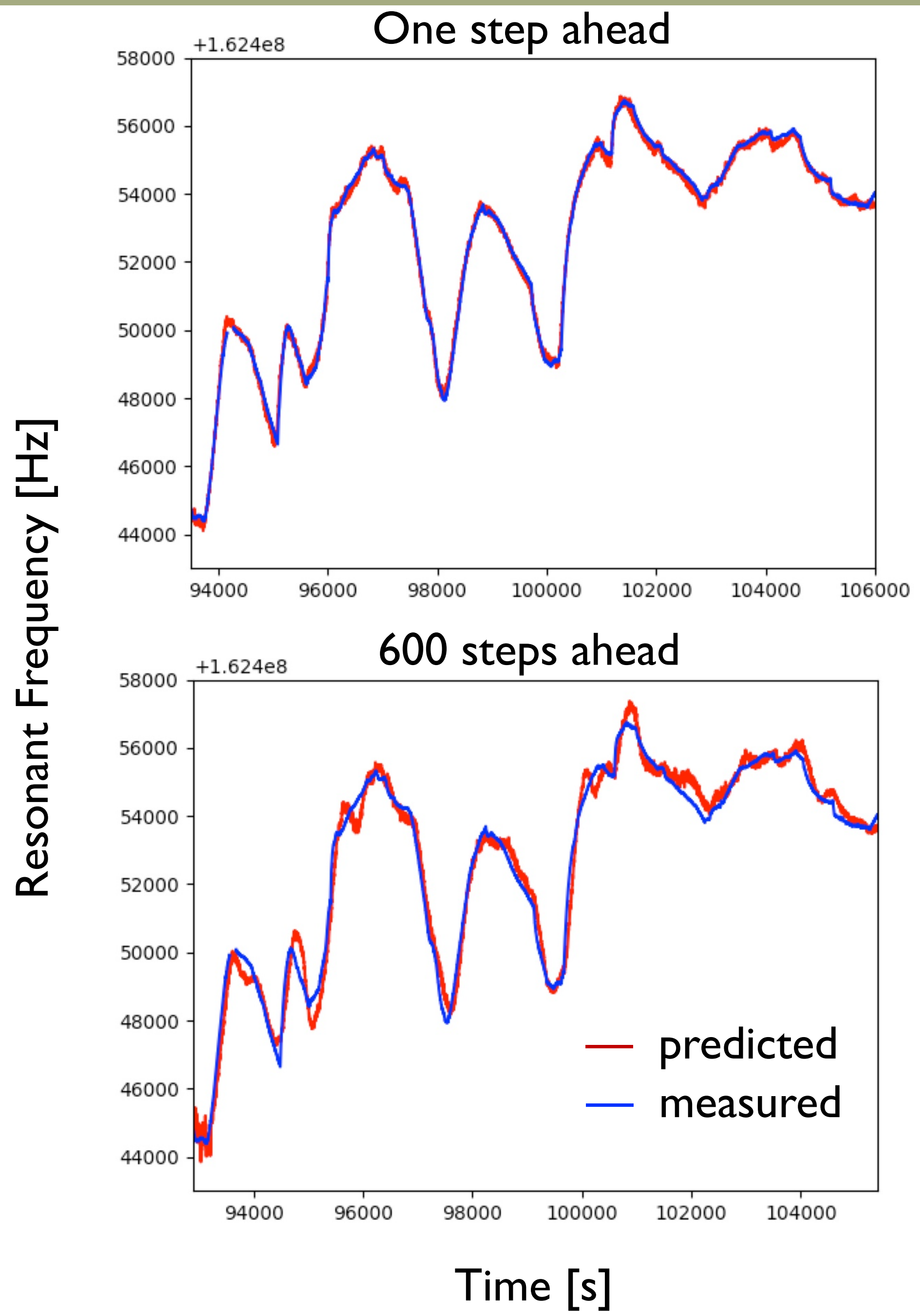
Other Stumbling Blocks…

- One-step ahead: 106 Hz MAE, 796 Hz max
- 600-step ahead: 339 Hz MAE, 1588 Hz max
- Found that MPC exploits the FF model quirks too much. Some options:
  - Do fewer time steps ahead and deal with longer control interval while looping through horizon → very clunky
  - Linearize around operating point as before → might as well use linear MPC
  - Restrict MPC options for valve settings more → lose ability to react quickly to trips



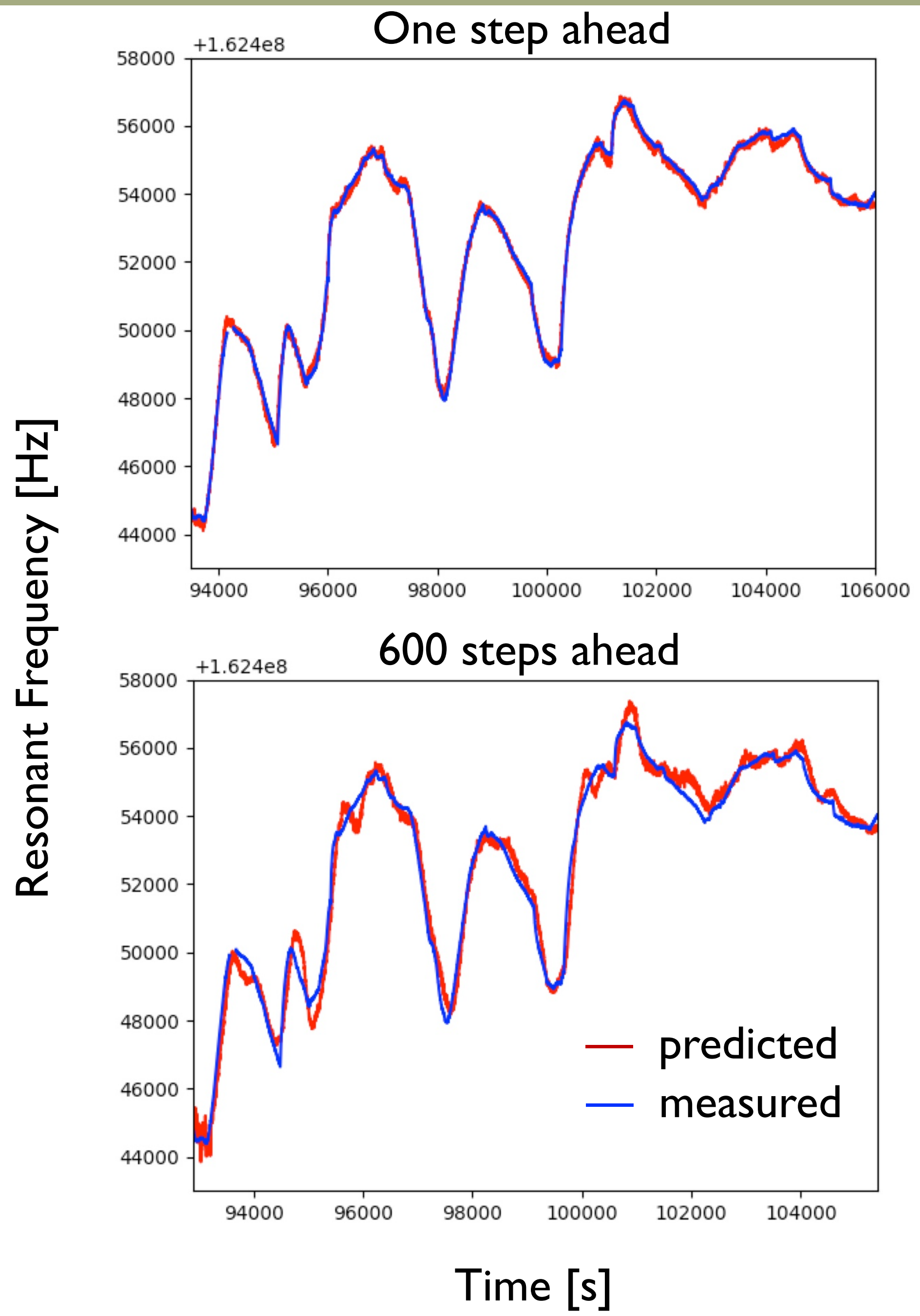
Other Stumbling Blocks…

- One-step ahead: 106 Hz MAE, 796 Hz max
- 600-step ahead: 339 Hz MAE, 1588 Hz max
- Found that MPC exploits the FF model quirks too much. Some options:
  - Do fewer time steps ahead and deal with longer control interval while looping through horizon → very clunky
  - Linearize around operating point as before → might as well use linear MPC
  - Restrict MPC options for valve settings more → lose ability to react quickly to trips
- Discovered that some of the early data taken during pulsed commissioning was bad (LLRF phase calibrations were not correct → had been re-set to wrong /old values!)



Other Stumbling Blocks…

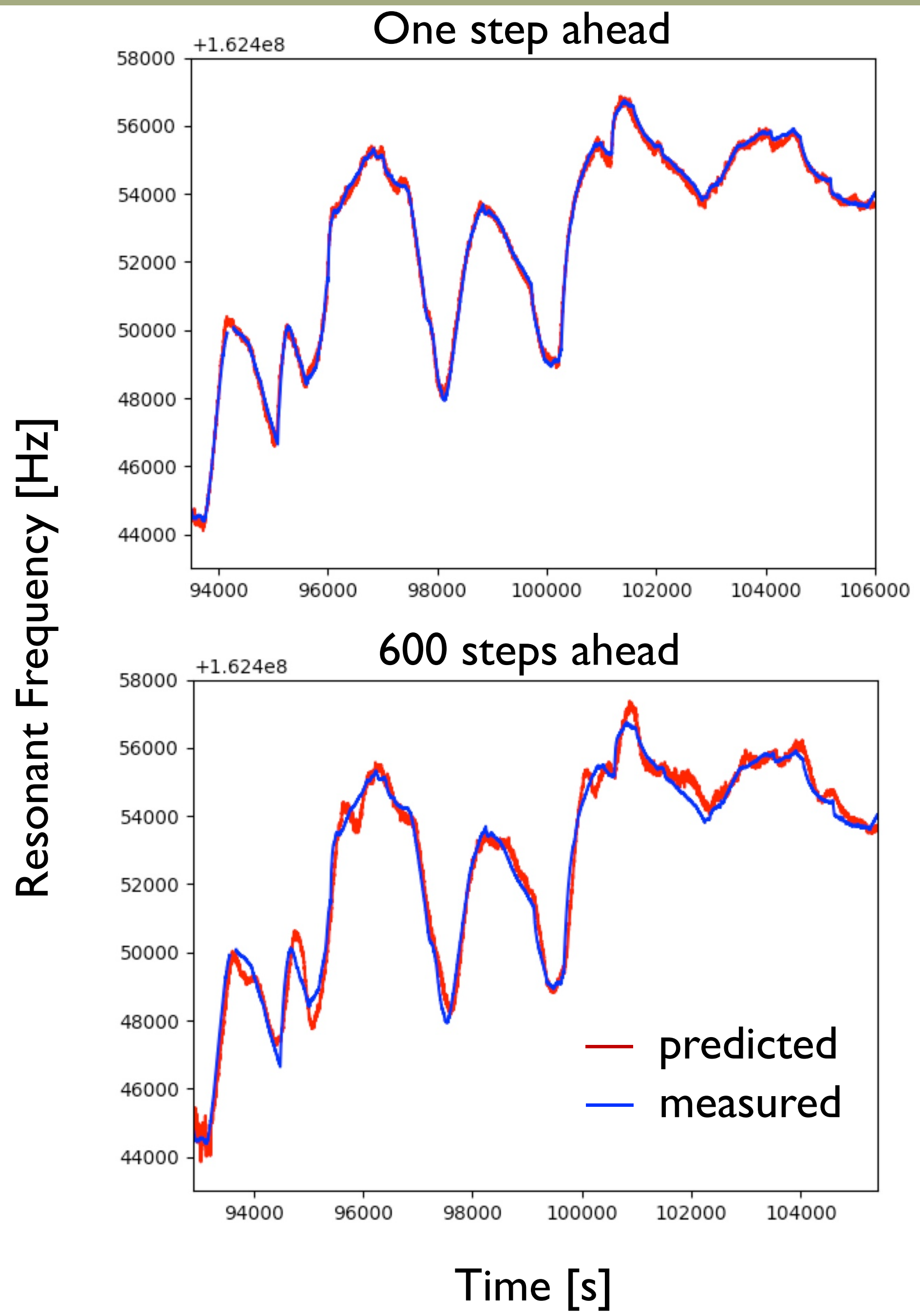
- One-step ahead: 106 Hz MAE, 796 Hz max
- 600-step ahead: 339 Hz MAE, 1588 Hz max
- Found that MPC exploits the FF model quirks too much. Some options:
  - Do fewer time steps ahead and deal with longer control interval while looping through horizon → very clunky
  - Linearize around operating point as before → might as well use linear MPC
  - Restrict MPC options for valve settings more → lose ability to react quickly to trips
- Discovered that some of the early data taken during pulsed commissioning was bad (LLRF phase calibrations were not correct → had been re-set to wrong /old values!)
- Discovered ambient humidity and temperature need to be explicitly predicted: ~1 kHz error reduction (vs. MPC standard of assuming constant)



Other Stumbling Blocks…

- One-step ahead: 106 Hz MAE, 796 Hz max
- 600-step ahead: 339 Hz MAE, 1588 Hz max
- Found that MPC exploits the FF model quirks too much. Some options:
  - Do fewer time steps ahead and deal with longer control interval while looping through horizon → very clunky
  - Linearize around operating point as before → might as well use linear MPC
  - Restrict MPC options for valve settings more → lose ability to react quickly to trips
- Discovered that some of the early data taken during pulsed commissioning was bad (LLRF phase calibrations were not correct → had been re-set to wrong /old values!)
- Discovered ambient humidity and temperature need to be explicitly predicted: ~1 kHz error reduction (vs. MPC standard of assuming constant)

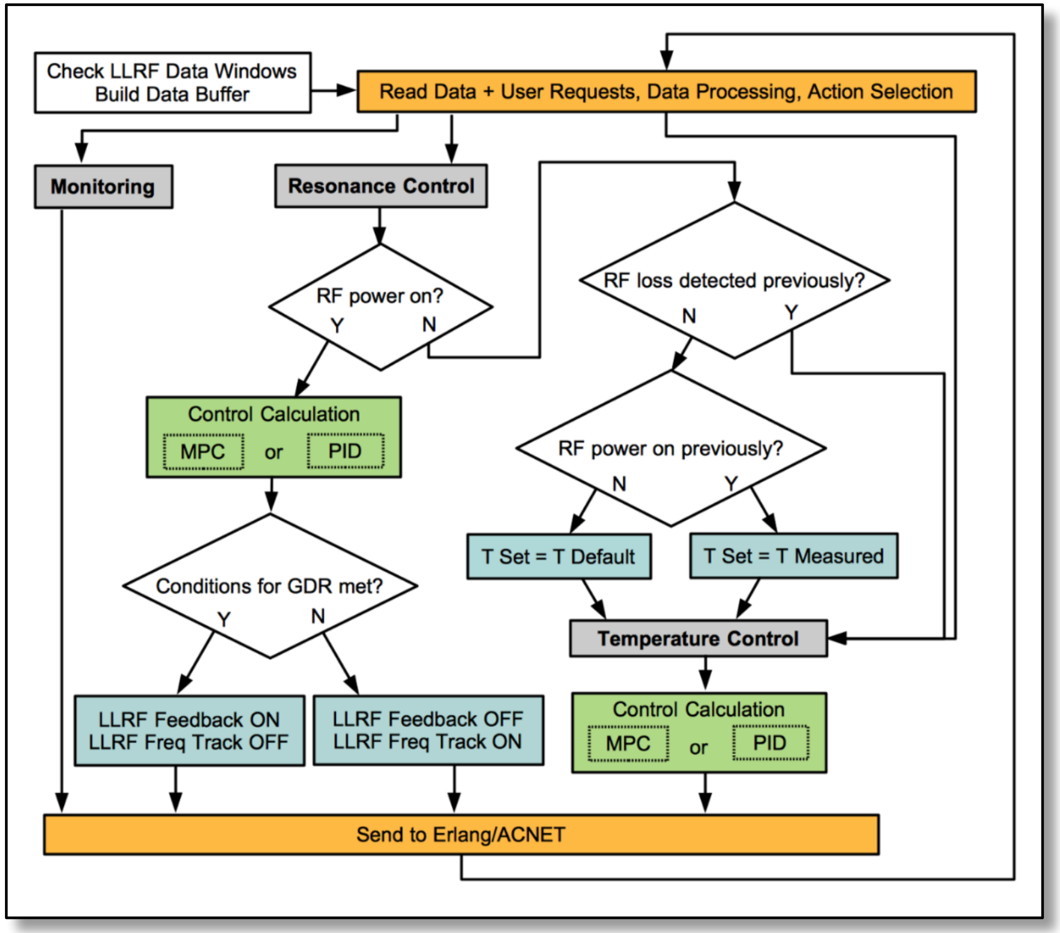
Decided to switch back to NN control policy approach

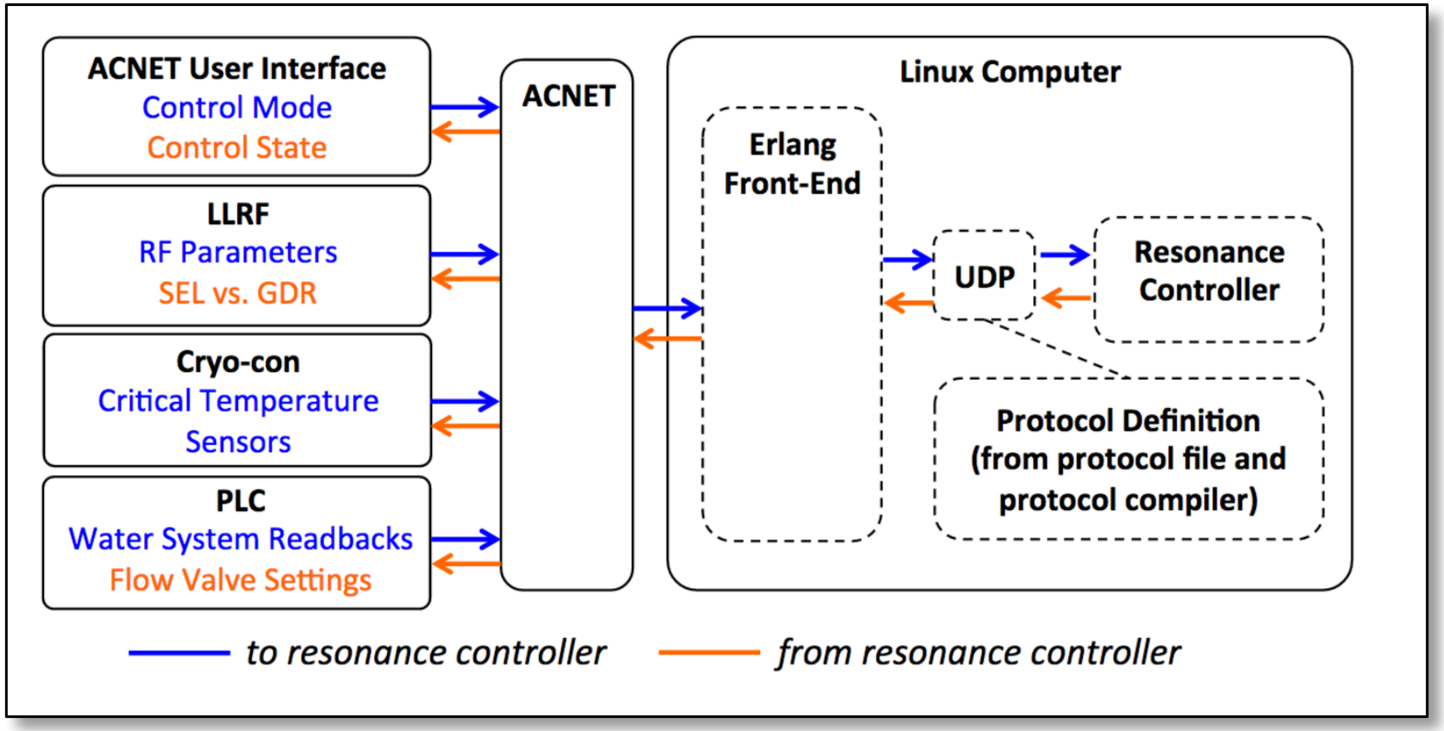


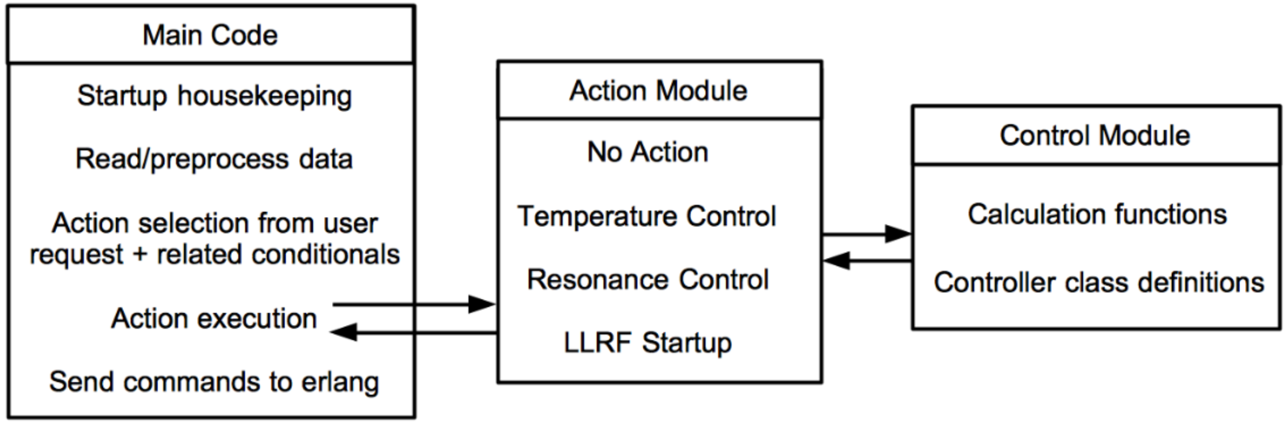
Even before that … had to actually put the infrastructure in place to use python with overarching lab control system (ACNET)

Built a python-based control framework

- *Executes on controls network linux computer*
- *PI control in regular operational use*
- *Designed to be portable + modular*
- *Supports the use of ML libraries*







Some lessons learned

- Model-based approaches require a lot of effort (ahead of any ML) → *payoff in terms of performance needs to be worth it to justify it*
- **"Simple" physics does not equal simple control/modeling!** → *esp. when one needs to take into account changes over time relative to control interval*
- Need appropriate infrastructure (and culture)
- Control systems deployment → *don't expect existing controls hardware/firmware to be up to the task for ML (especially for old facilities)*

Another set of applications: fast switching between operating conditions

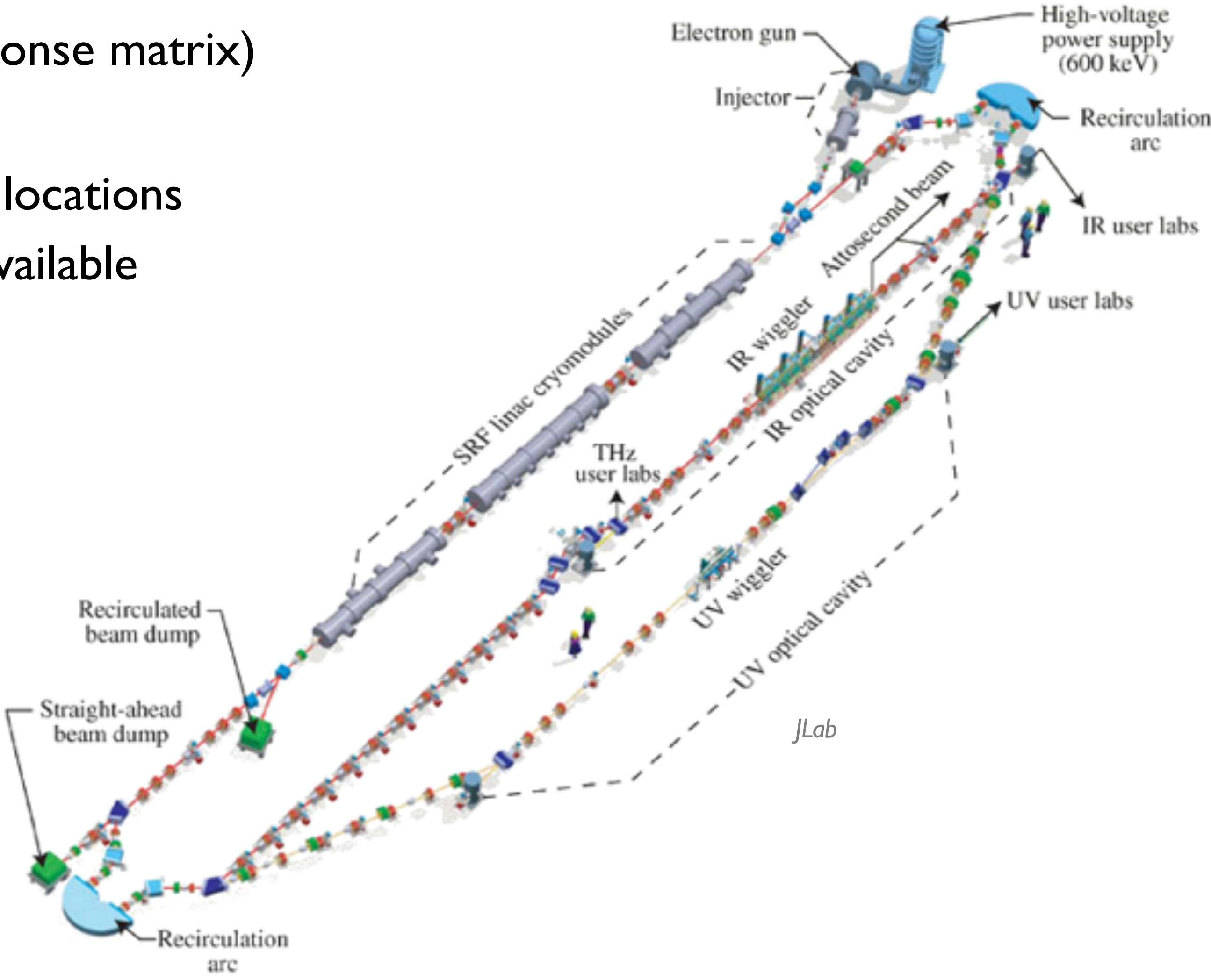
Fast Switching Between Trajectories

Work with C. Tennant and D. Douglas, JLab

- 76 BPMs, 57 dipoles, 53 quadrupoles
- Traditional approach has never worked (linear response matrix)
- Rely on one expert for steering tune-up
- Want to specify small offsets in trajectory at some locations
- Didn't initially have an up-to-date machine model available

Learn responses (NN model) from tune-up data and dedicated study time:
dipole + quadrupole settings → predict BPMs

Train controller (NN policy) offline using NN model: desired trajectory + present settings + BPM readbacks → change in dipole settings
(and penalize losses + large magnet settings)



JLab

Fast Switching Between Trajectories

- 76 BPMs, 57 dipoles, 53 quadrupoles
- Traditional approach has never worked (linear response matrix)
- Rely on one expert for steering tune-up
- Want to specify small offsets in trajectory at some locations
- Didn't initially have an up-to-date machine model available

Learn responses (NN model) from tune-up data and dedicated study time:
dipole + quadrupole settings → predict BPMs

Train controller (NN policy) offline using NN model: desired trajectory + present settings + BPM readbacks → change in dipole settings
(and penalize losses + large magnet settings)

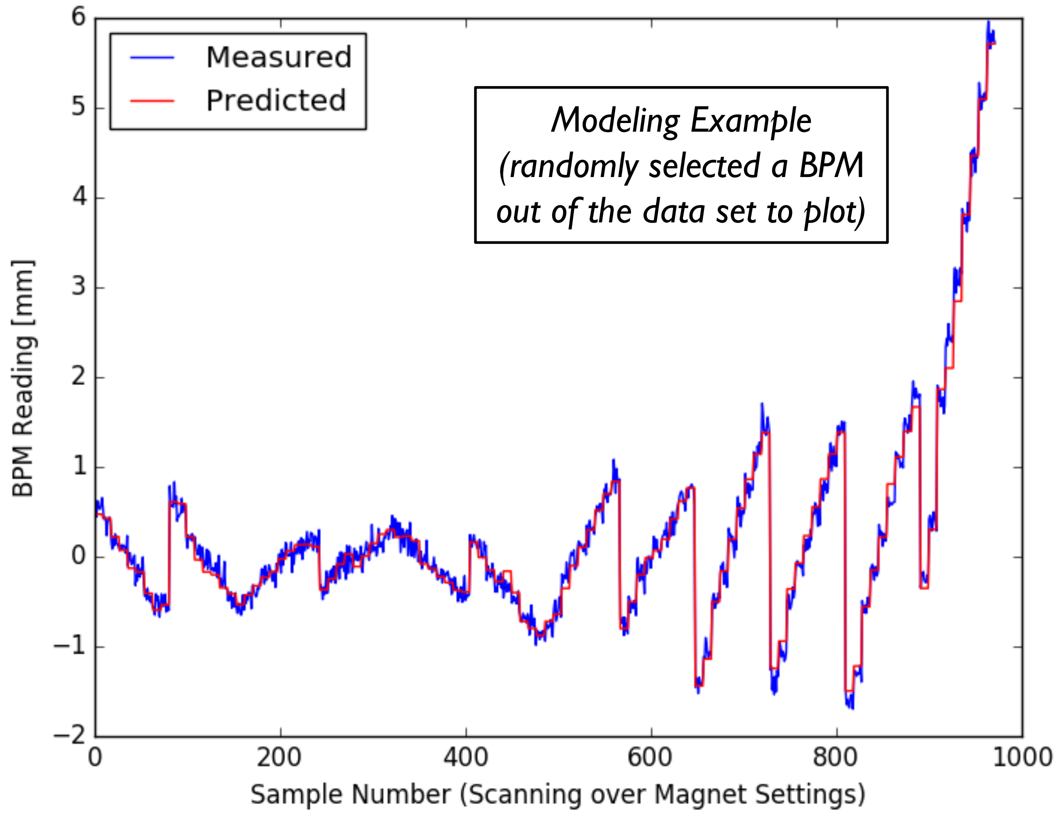
(Very) Preliminary Results:

Model Errors for BPMs:

Training Set:	0.07 mm MAE	0.09 mm STD
Validation Set:	0.08 mm MAE	0.07 mm STD
Test Set:	0.08 mm MAE	0.03 mm STD

Controller:

random initial states → on average within 0.2 mm of center immediately using 8 dipoles



Fast Switching Between Trajectories

Example of learning machine model
from measured data alone *(including tune-up data)*

But what about a machine test?

Fast Switching Between Trajectories

Example of learning machine model
from measured data alone *(including tune-up data)*

But what about a machine test?

Started in 2012 → machine shut down 6 months later

- *Short run (several weeks) in 2016 to gather data after substantial machine changes*
- *Unlikely to turn on again to be able to test*

Do have an ok model in *elegant* now:

- *Still have mismatch, but can test adapting to new conditions*

Fast Switching Between Trajectories

Example of learning machine model

from measured data alone *(including tune-up data)*

But what about a machine test?

Started in 2012 → machine shut down 6 months later

- *Short run (several weeks) in 2016 to gather data after substantial machine changes*
- *Unlikely to turn on again to be able to test*

Do have an ok model in *elegant* now:

- *Still have mismatch, but can test adapting to new conditions*

Comparisons with standard approach?
(integral feedback with inverted linear response matrix)

Main possible advantage of NN over standard approach:

- *Adaptive control policy → can adjust without interfering with operation for response measurements as often*
- *Handling of trajectories away from BPM center (nonlinear)*
- ***But, need to quantify this …***

Simulation study: switching between beam energies for a compact FEL

Would be nice to have a tool that can quickly give suggested settings for a given photon beam request, is valid globally, and can adapt to changes over time

Motivation: Switching Between User Requests in FELs

- FEL facilities support a wide variety of scientific endeavors *(e.g. imaging protein structures[1], understanding processes like photosynthesis[2], origin of material properties[3])*
- Need to accommodate requests for a wide variety of photon beam characteristics
- May switch as often as every few days
- Have save/restore settings, but these are discrete, and there can be some drift in the machine
- Time spent tuning = reduced scientific output for a given operational budget

e.g. *the Linac Coherent Light Source*

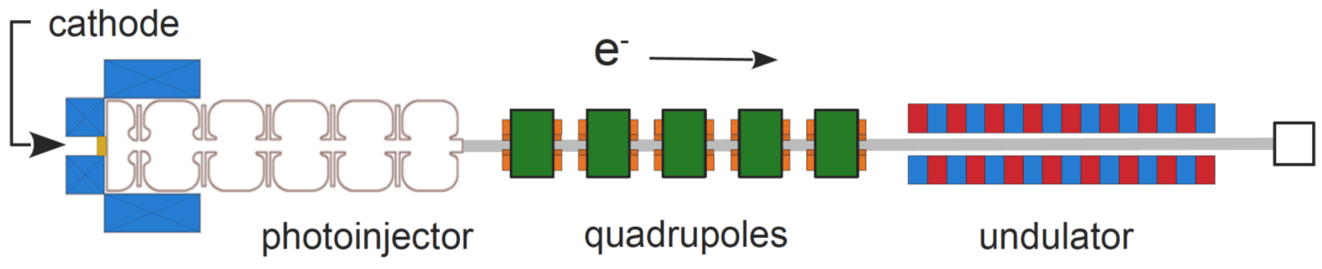
(image: lcls.slac.standford.edu)

Would be nice to have a tool that can quickly give suggested settings for a given photon beam request, is valid globally, and can adapt to changes over time

[1] J.-P. Colletier, et al.,"De novo phasing with X-ray laser reveals mosquito larvicide BinAB structure," Nature , vol. 539, pp. 43–47, Sep. 2016.
[2] I. D. Young, et al., "Structure of photosystem II and substrate binding at room temperature," Nature , vol. 540, pp. 453–457, Nov. 2016.
[3] M. P. Jiang, et al., "The origin of incipient ferroelectricity in lead telluride," Nature Communications, vol. 7, no. 12291, Jul. 2016.

Starting Smaller: A Case Study

Compact, THz FEL design based on previously operational TEU-FEL



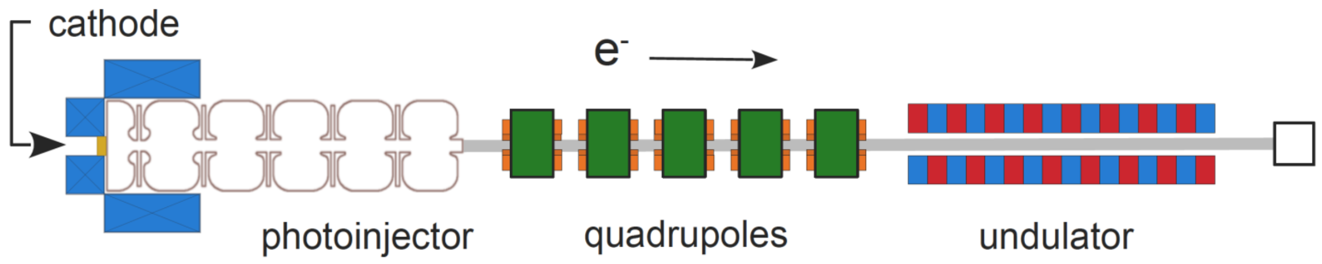
3 – 6 MeV electron beam
200 – 800 μm photon beam

Previously operated at University of Twente in the Netherlands

Was going to be re-built at CSU: have simulation from design studies

Starting Smaller: A Case Study

Compact, THz FEL design based on previously operational TEU-FEL



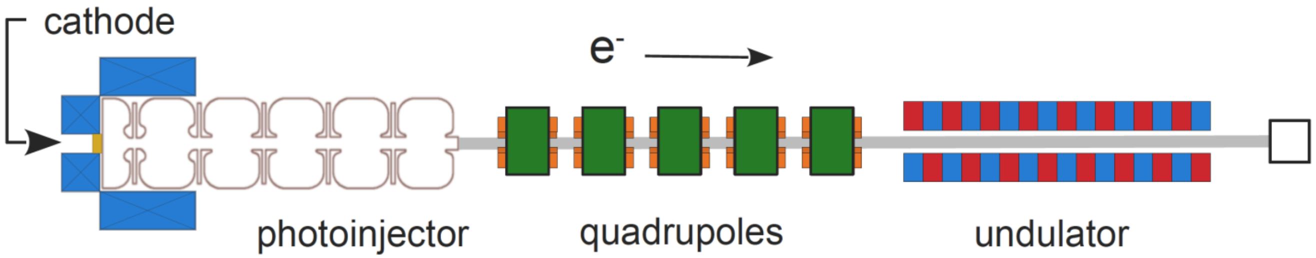
3 – 6 MeV electron beam
200 – 800 μm photon beam

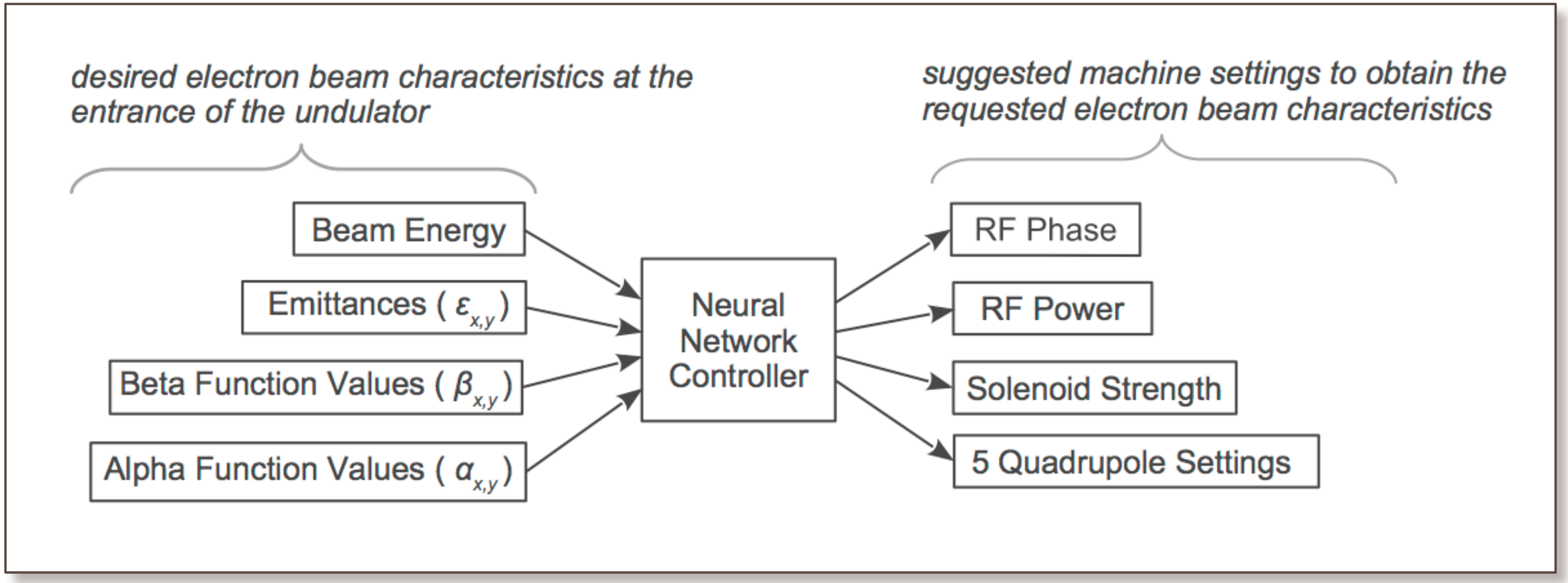
Previously operated at University of Twente in the Netherlands

Was going to be re-built at CSU: have simulation from design studies

This is an appealing system for an initial study because it has a small number of machine components, yet it exhibits non-trivial beam dynamics.

Intermediate goal: get the right beam parameters at the undulator entrance





First: Learn a Model from Simulation Results

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

Decided to learn a neural network model from simulation:

- *faster-executing than physics-based simulation*
- *can update with measured data*

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

Decided to learn a neural network model from simulation:

- *faster-executing than physics-based simulation*
- *can update with measured data*

More broadly: machine time is expensive, mistakes can be costly, and simulations don't always match the machine well

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

Decided to learn a neural network model from simulation:

- *faster-executing than physics-based simulation*
- *can update with measured data*

More broadly: machine time is expensive, mistakes can be costly, and simulations don't always match the machine well

→ *Sample efficiency matters a lot (both with slow sim and machine)*

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

Decided to learn a neural network model from simulation:

- *faster-executing than physics-based simulation*
- *can update with measured data*

More broadly: machine time is expensive, mistakes can be costly, and simulations don't always match the machine well

→ *Sample efficiency matters a lot (both with slow sim and machine)*

→ *Learning a machine model using simulation results and updating it with existing measurements can aid controller training*

First: Learn a Model from Simulation Results

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes space charge (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → *couldn't just run a lot of interactions with controller on a cluster*

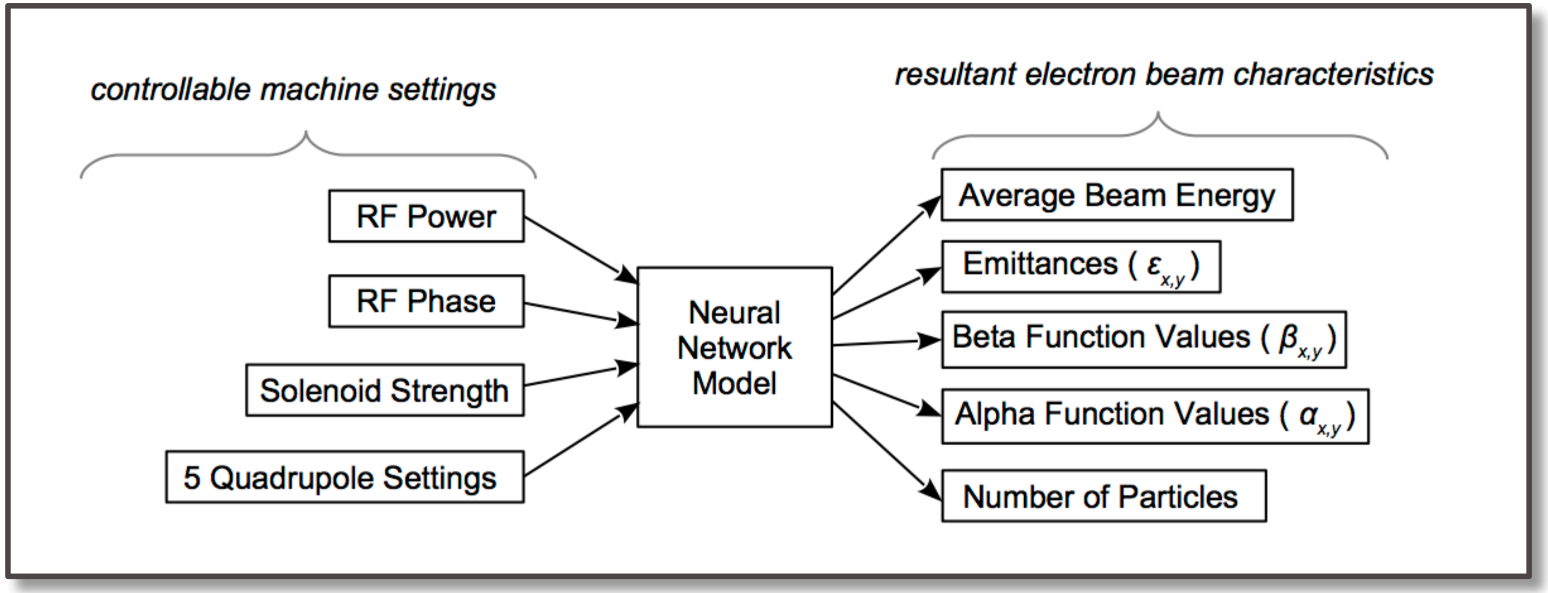
Decided to learn a neural network model from simulation:

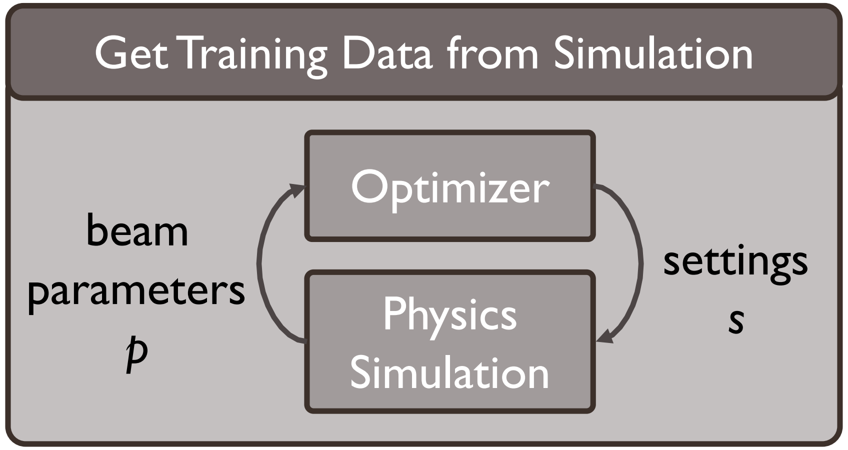
- *faster-executing than physics-based simulation*
- *can update with measured data*

More broadly: machine time is expensive, mistakes can be costly, and simulations don't always match the machine well

→ *Sample efficiency matters a lot (both with slow sim and machine)*

→ *Learning a machine model using simulation results and updating it with existing measurements can aid controller training*

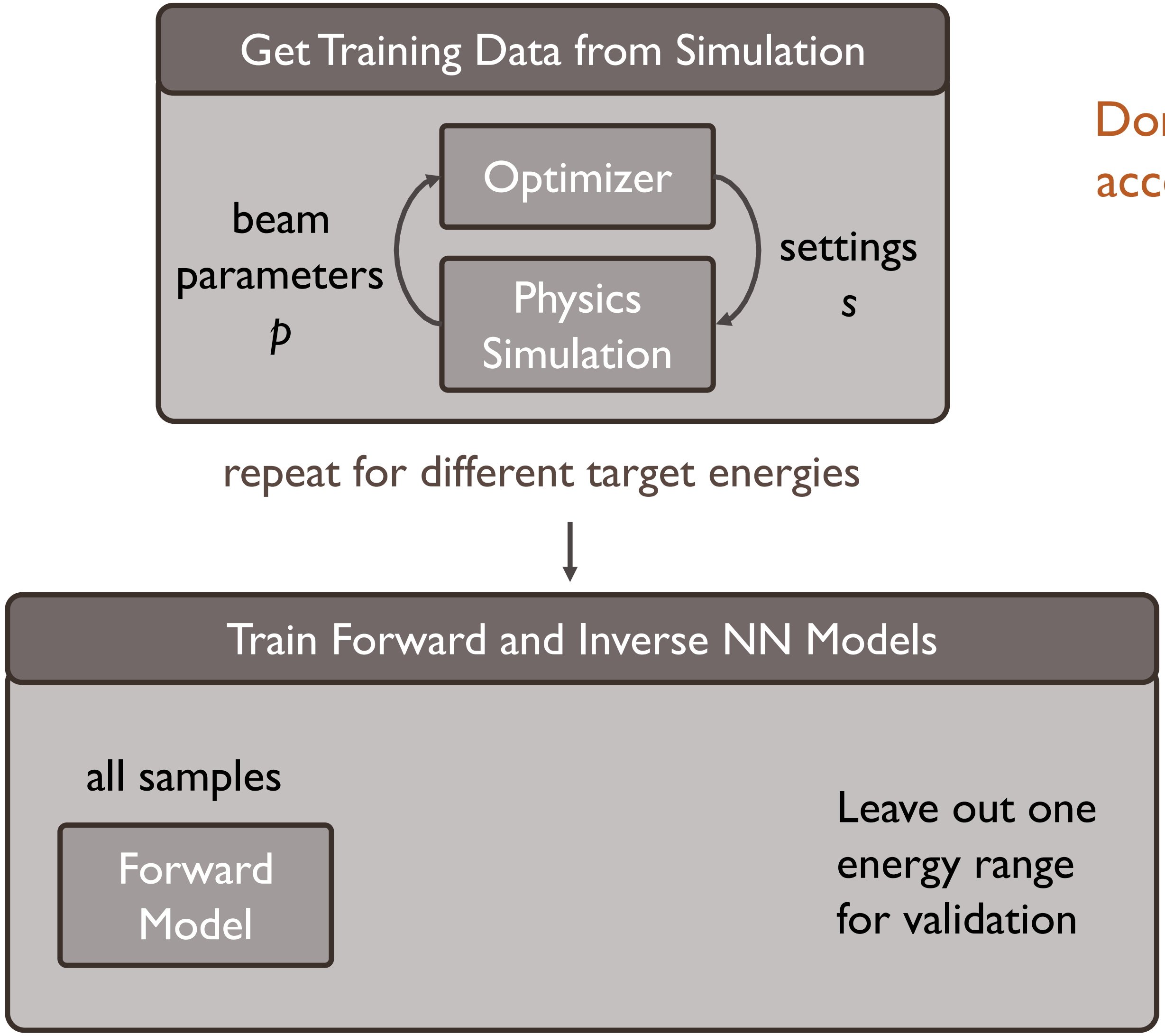




repeat for different target energies

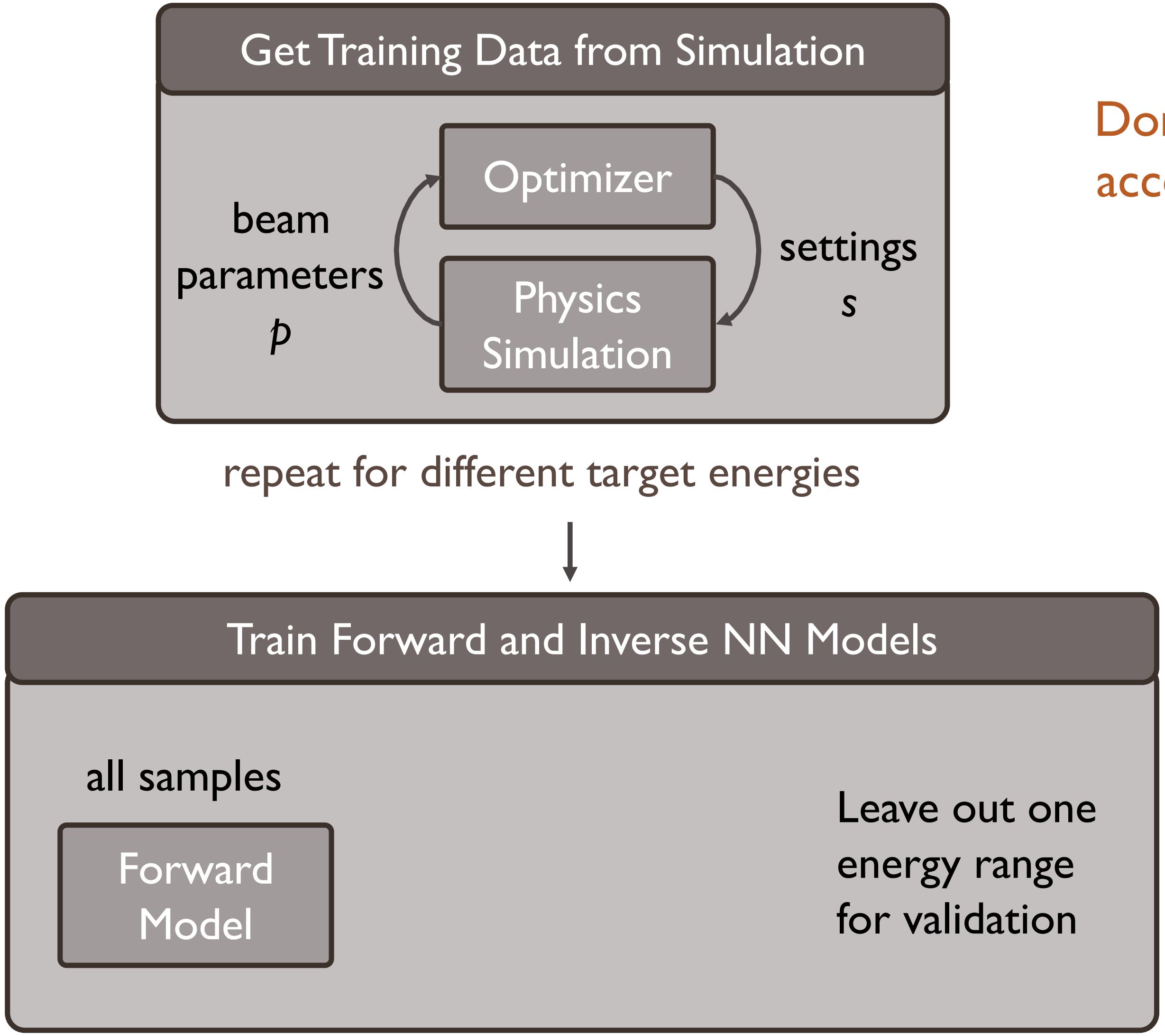
Don't always have a good physics-based model for particle accelerators, so what's in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings



Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility?

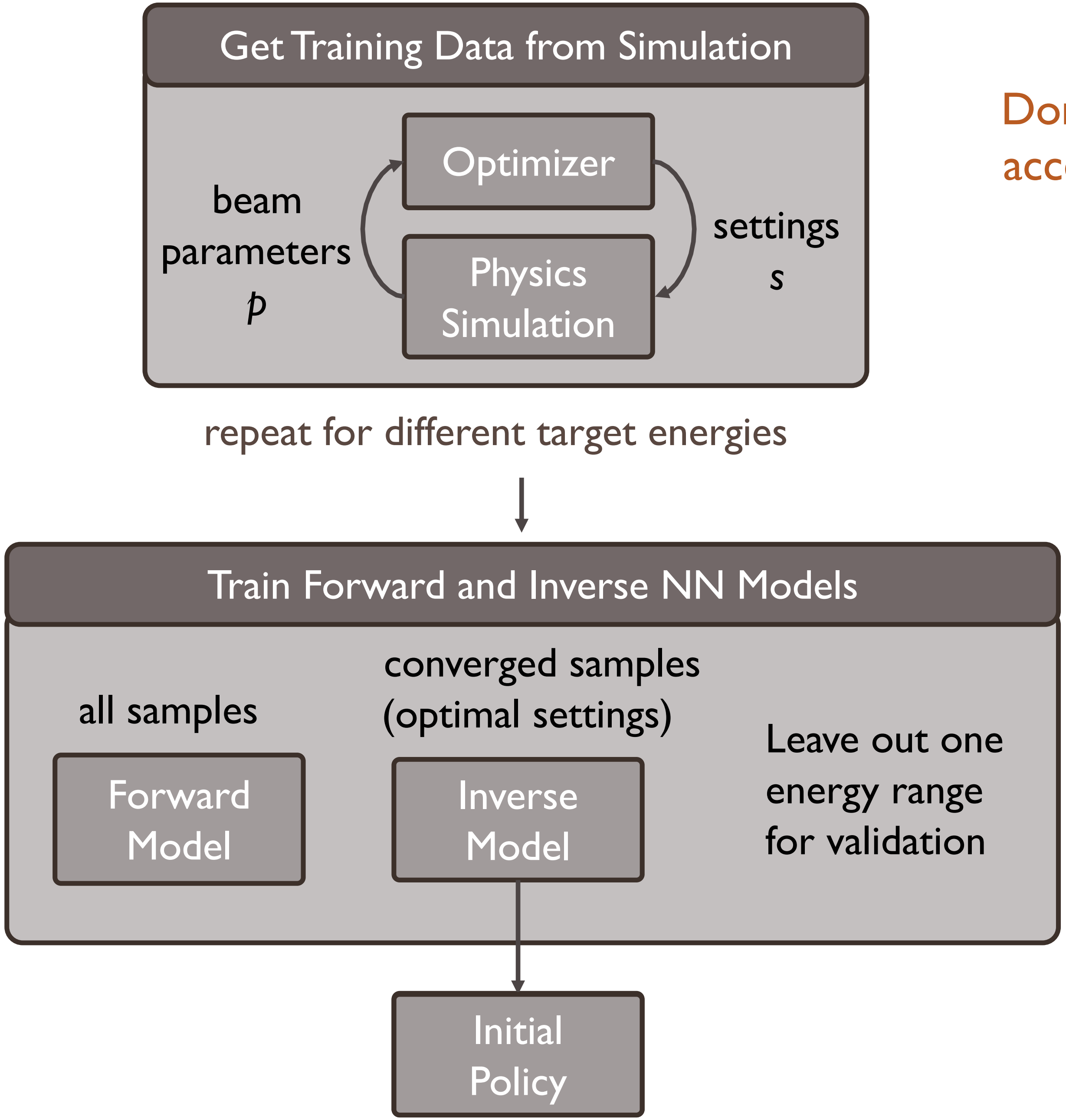
Noisy data + tuning around roughly optimal settings



Don't always have a good physics-based model for particle accelerators, so what's in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings

Want to use the existing data to initialize control policy



Don't always have a good physics-based model for particle accelerators, so what's in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings

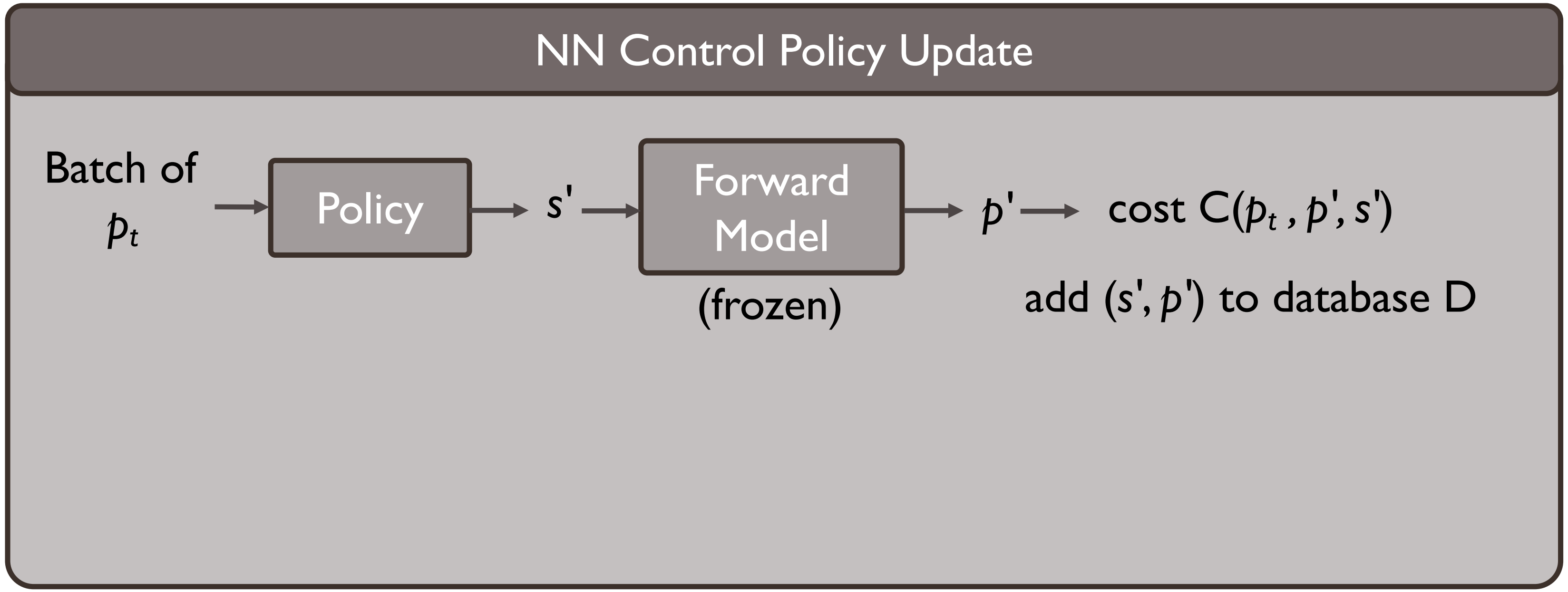
Want to use the existing data to initialize control policy → *model not invertible, but can pre-train policy with converged settings*

Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*

Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*



p_t -- target beam parameters

s' -- predicted optimal settings

p' – predicted beam parameters

Cost:
difference between p' and p_t
penalize loss of transmission
penalize higher magnet settings

Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*

NN Control Policy Update

Batch of p_t → Policy → s' → Forward Model (frozen) → p' → cost $C(p_t, p', s')$

add (s', p') to database D

Every n^th iteration, take batch of s', p' sampled from D, run through physics simulation, and update the model

p_t -- target beam parameters

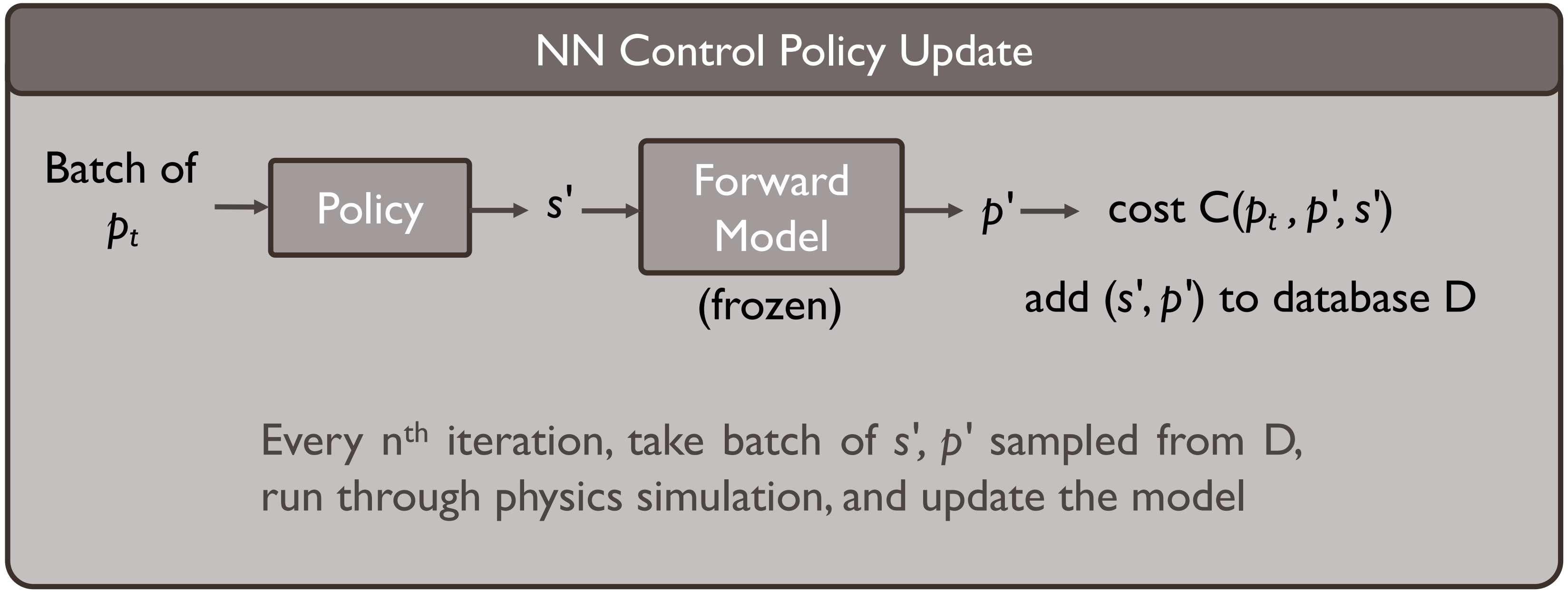
s' -- predicted optimal settings

p' – predicted beam parameters

Cost:
difference between p' and p_t
penalize loss of transmission
penalize higher magnet settings

Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*



p_t -- target beam parameters

s' -- predicted optimal settings

p' – predicted beam parameters

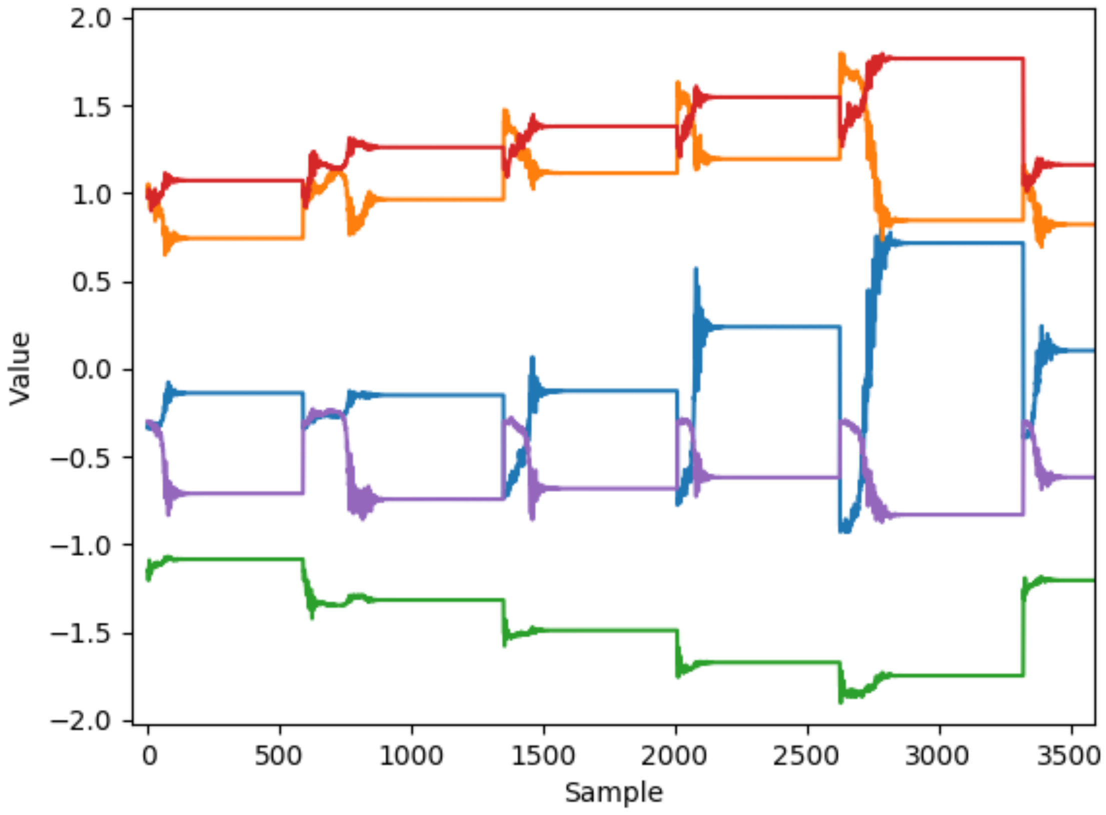
Cost:
difference between p' and p_t
penalize loss of transmission
penalize higher magnet settings

Then test policy directly on simulation

Initial Model and Policy

Training data from simulation:

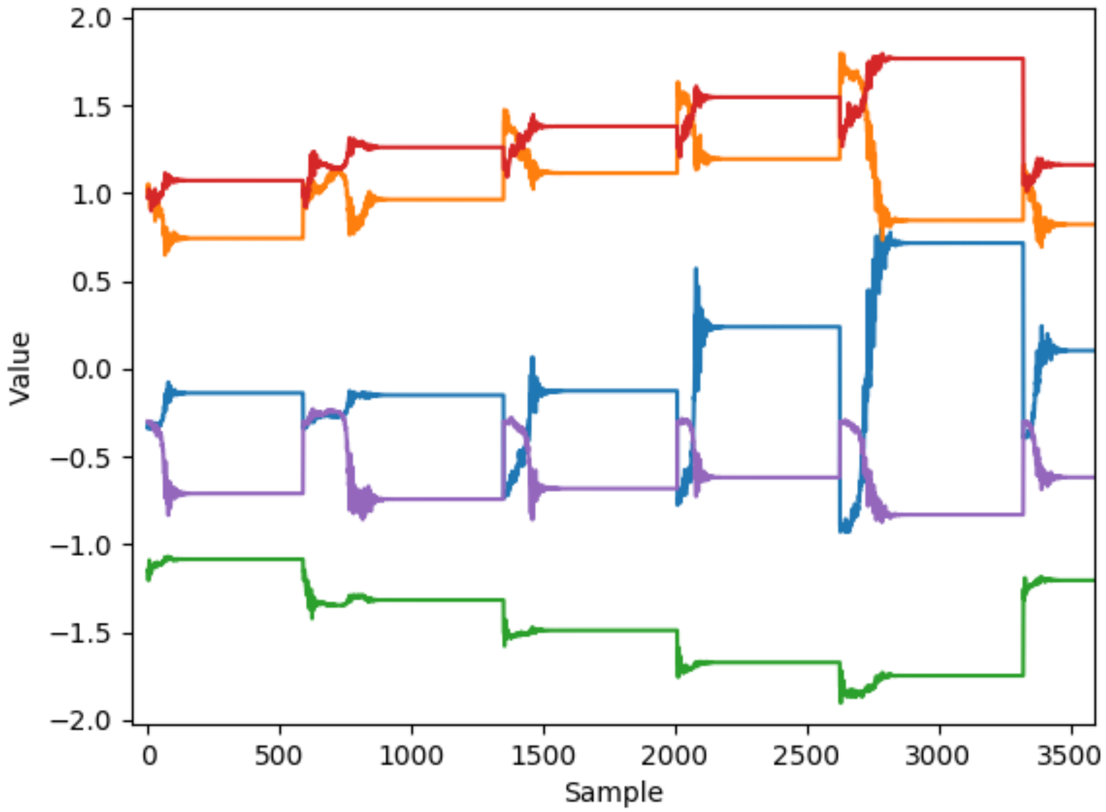
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)



Example of what the training data looks like (quadrupoles shown in this case)

Initial Model and Policy

Training data from simulation:
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)



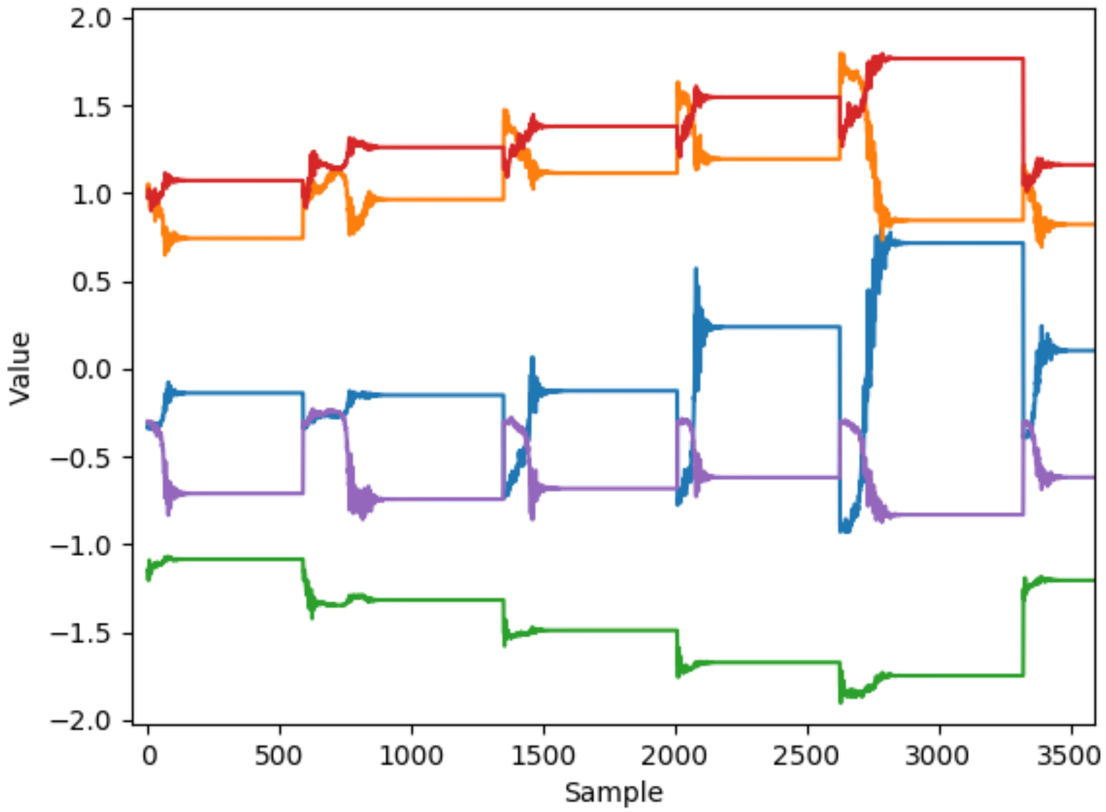
Example of what the training data looks like (quadrupoles shown in this case)

Model: 50-50-30-30 tanh nodes in hidden layers
- 8 inputs *(rf power, rf phase, sol. strength, quads)*
- 8 outputs (α_x, α_y, β_x, β_y, ε_x, ε_y, E, N_p)
- 5.7-MeV run used for validation set

Initial Model and Policy

Training data from simulation:

- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)



Example of what the training data looks like (quadrupoles shown in this case)

Model: 50-50-30-30 tanh nodes in hidden layers

- 8 inputs *(rf power, rf phase, sol. strength, quads)*
- 8 outputs (α_x , α_y , β_x , β_y , ε_x , ε_y , E , N_p)
- 5.7-MeV run used for validation set

First study: focus on target α, β for a given energy

Policy: 30-30-20-20 tanh nodes in hidden layers

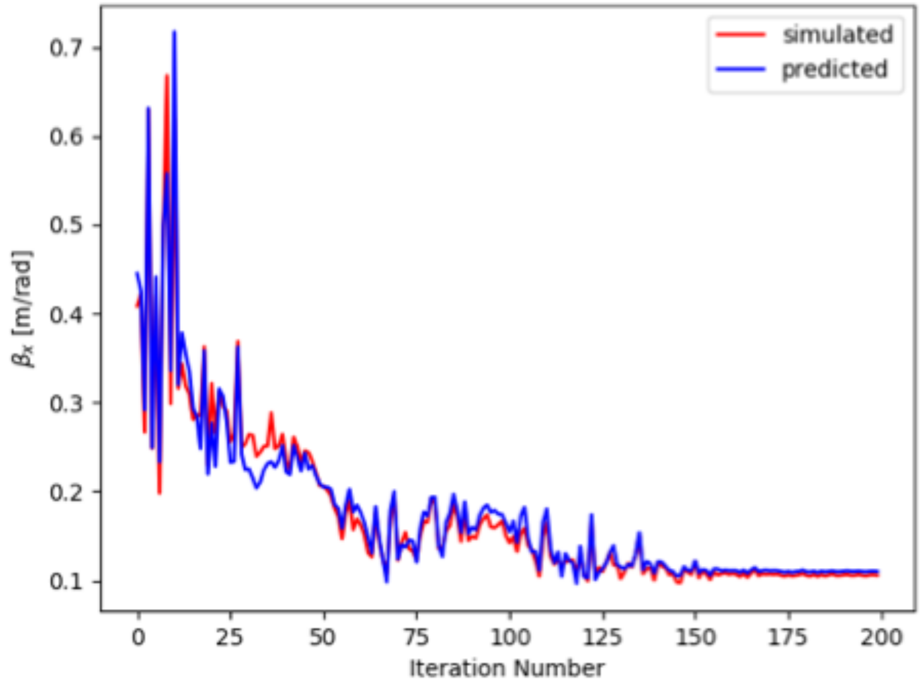
- inputs/outputs opposite the above (except N_p)
- random target energies, $\alpha_{xy} = 0, \beta_{xy} = 0.106$
- exclude 4.8 – 5.2 MeV range for validation

Initial Model and Policy Performance

Summary of Model Performance

Parameter	Train MAE	Train STD	Train Max	Val. MAE	Val. STD	Val. Max
α_x [rad]	0.018	0.042	0.590	0.067	0.091	0.482
α_y [rad]	0.022	0.037	0.845	0.070	0.079	0.345
β_x [m/rad]	0.004	0.009	0.287	0.008	0.012	0.130
β_y [m/rad]	0.005	0.011	0.357	0.012	0.017	0.189

Example of Model Performance on Validation Set



Initial Model and Policy Performance

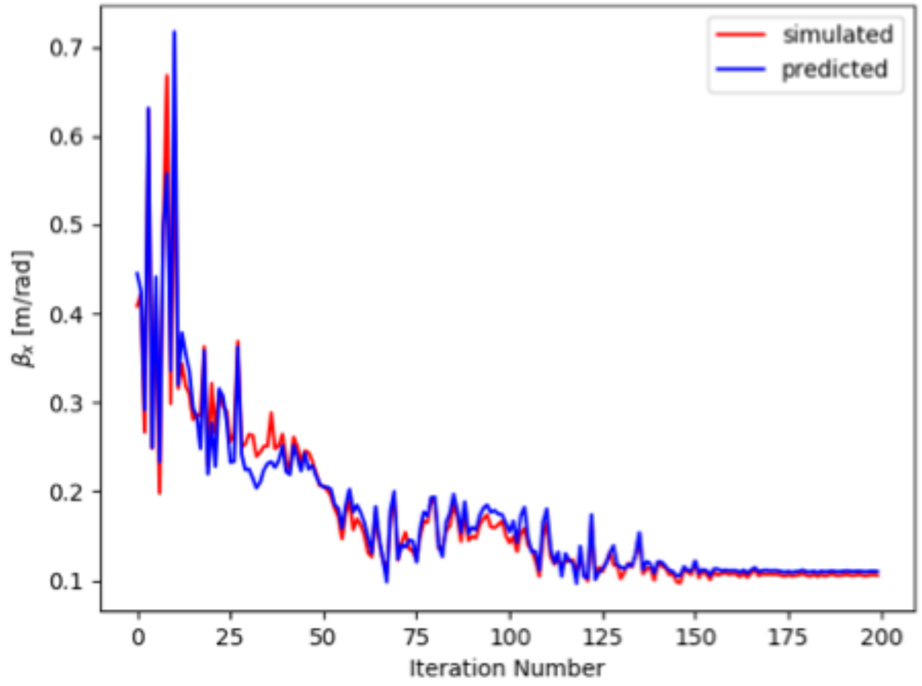
Summary of Model Performance

Parameter	Train MAE	Train STD	Train Max	Val. MAE	Val. STD	Val. Max
α_x [rad]	0.018	0.042	0.590	0.067	0.091	0.482
α_y [rad]	0.022	0.037	0.845	0.070	0.079	0.345
β_x [m/rad]	0.004	0.009	0.287	0.008	0.012	0.130
β_y [m/rad]	0.005	0.011	0.357	0.012	0.017	0.189

Controller ability to reach $\alpha_{x,y}$ = 0 and $\beta_{x,y}$ = 0.106 in **one iteration**

Parameter	Train MAE	Train STD	Train Max	Val. MAE	Val. STD	Val. Max
α_x [rad]	0.012	0.075	0.011	0.046	0.063	0.141
α_y [rad]	0.013	0.079	0.012	0.045	0.064	0.140
β_x [m/rad]	0.008	0.004	0.006	0.006	0.023	0.008
β_y [m/rad]	0.014	0.011	0.011	0.011	0.069	0.038

Example of Model Performance on Validation Set

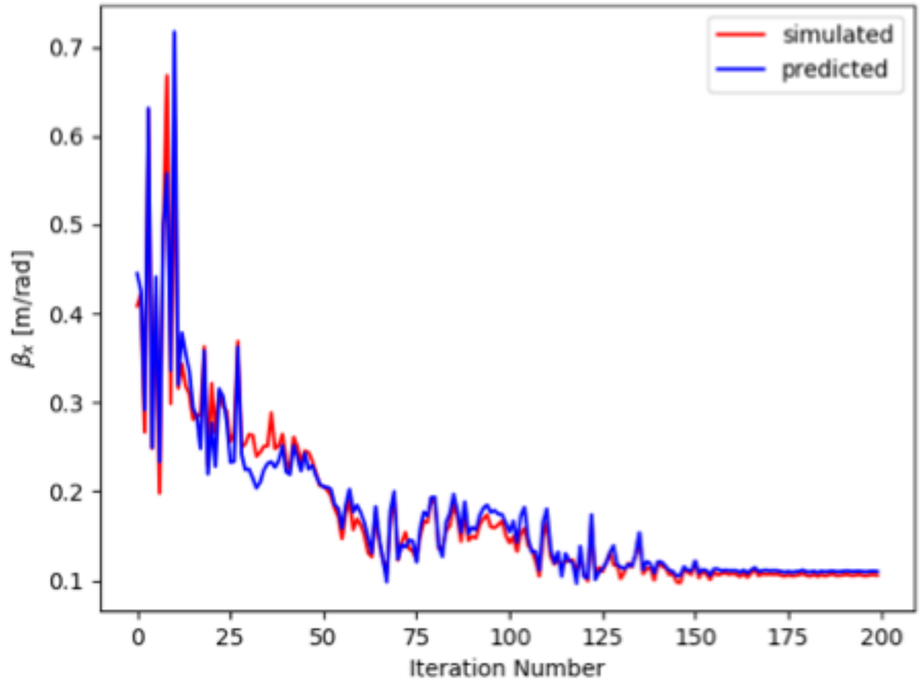


Initial Model and Policy Performance

Summary of Model Performance

Parameter	Train MAE	Train STD	Train Max	Val. MAE	Val. STD	Val. Max
α_x [rad]	0.018	0.042	0.590	0.067	0.091	0.482
α_y [rad]	0.022	0.037	0.845	0.070	0.079	0.345
β_x [m/rad]	0.004	0.009	0.287	0.008	0.012	0.130
β_y [m/rad]	0.005	0.011	0.357	0.012	0.017	0.189

Example of Model Performance on Validation Set



Controller ability to reach $\alpha_{x,y}$ = 0 and $\beta_{x,y}$ = 0.106 in **one iteration**

Parameter	Train MAE	Train STD	Train Max	Val. MAE	Val. STD	Val. Max
α_x [rad]	0.012	0.075	0.011	0.046	0.063	0.141
α_y [rad]	0.013	0.079	0.012	0.045	0.064	0.140
β_x [m/rad]	0.008	0.004	0.006	0.006	0.023	0.008
β_y [m/rad]	0.014	0.011	0.011	0.011	0.069	0.038

What this means: for a given energy, the controller will immediately reach the desired beam size to within about 10% and the beam will be close to a waist, requiring minimal further tuning (assuming no substantial drift…)

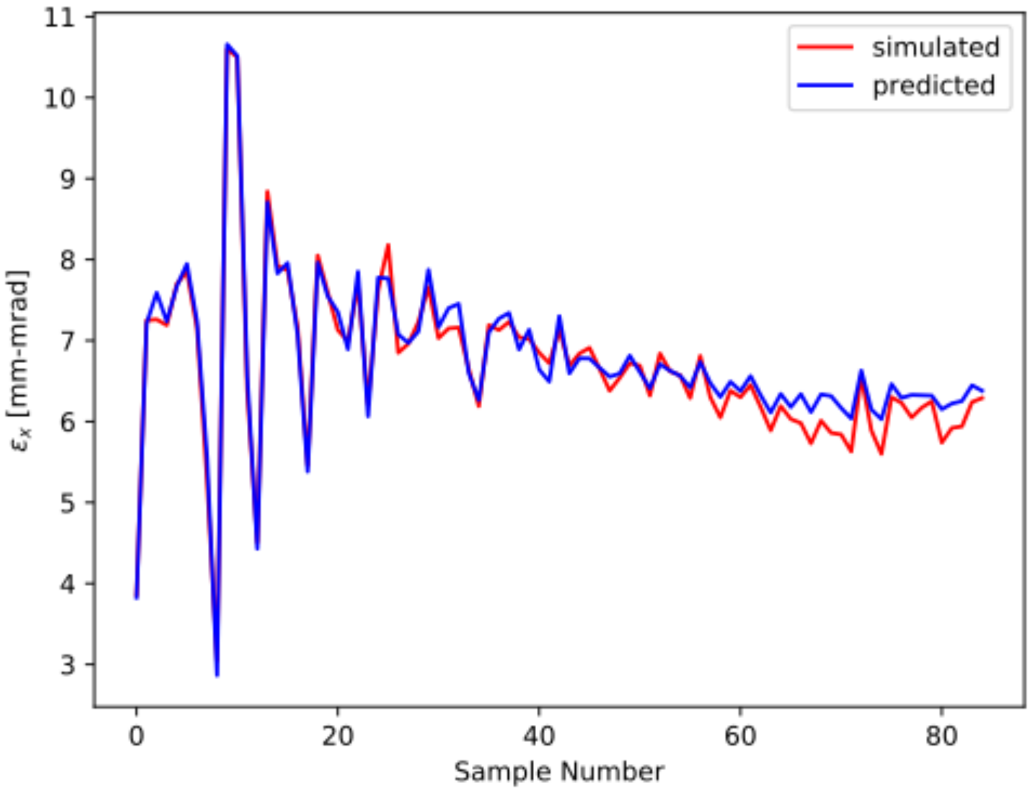
A.L. Edelen et al., FEL '17

Presently finishing more complete study

- Including minimization of emittance + more freedom with injector settings
  - *Requires finer start-to-end adjustments, so more simulation data was needed*
  - *Much larger network needed to capture relationships accurately in model*
- Seeing how well it does with machine drift
  - *e.g. deviation between settings and real values, deviation in responses*
- Other changes to setup
  - *More standard RL*
- So far, only showed results for the electron beam
- Need to compare with other methods
  - *Esp. model-free RL methods, traditional online optimization*

The effort of model creation may not scale well to larger facilities relative to performance gain

Example of Model Performance on Validation Set



Some Practical Challenges

*large enough parameter range and set of examples to generalize well and complete the task

*esp. consistent!

Need a sufficient amount of reliable* data*

(but not as much as is sometimes claimed in DL)

Training on Measured Data

Undocumented manual changes
(e.g. rotating a BPM, Quad)

Relevant-but-unlogged variables

Availability of diagnostics
(old machines, camera servers, machine subsections)

Observed parameter range in archived data

Time on machine for characterization studies
(schedule + expense)

Ideal case:
- *comprehensive, high-resolution data archive (e.g. including things like ambient temp./pressure)*
- *excellent log of manual changes*

Some Practical Challenges

*large enough parameter range and set of examples to generalize well and complete the task

*esp. consistent!

Need a sufficient amount of reliable* data*

(but not as much as is sometimes claimed in DL)

Training on Measured Data

Undocumented manual changes
(e.g. rotating a BPM, Quad)

Relevant-but-unlogged variables

Availability of diagnostics
(old machines, camera servers, machine subsections)

Observed parameter range in archived data

Time on machine for characterization studies
(schedule + expense)

Ideal case:
- *comprehensive, high-resolution data archive (e.g. including things like ambient temp./pressure)*
- *excellent log of manual changes*

Training on Simulation Data

How representative of the real machine behavior?

Input/output parameters need to translate directly to what's on the machine (quantitatively) — need coordination up front

High-fidelity (e.g. PIC)
→ time-consuming to run

Retention + availability of prior results:
(optimize and throw the iterations away!)

Some Practical Challenges

*large enough parameter range and set of examples to generalize well and complete the task

*esp. consistent!

Need a sufficient amount of reliable* data*

(but not as much as is sometimes claimed in DL)

Training on Measured Data

Undocumented manual changes
(e.g. rotating a BPM, Quad)

Relevant-but-unlogged variables

Availability of diagnostics
(old machines, camera servers, machine subsections)

Observed parameter range in archived data

Time on machine for characterization studies
(schedule + expense)

Ideal case:
- *comprehensive, high-resolution data archive (e.g. including things like ambient temp./pressure)*
- *excellent log of manual changes*

Training on Simulation Data

How representative of the real machine behavior?

Input/output parameters need to translate directly to what's on the machine (quantitatively) — need coordination up front

High-fidelity (e.g. PIC) → time-consuming to run

Retention + availability of prior results:
(optimize and throw the iterations away!)

Deployment

Initial training is on HPC systems → deployment is typically not*
- Execution on front-end: necessary speed + memory?
- Subsequent training: on front-end or transfer to HPC?

Software compatibility for older systems:
interface with machine + make use of modern ML software libraries

I/O for large amounts of data

* for now…

Final Notes

- Neural networks are very flexible tools → far more powerful in recent years
- Most of the real work comes before the actual ML …
- Mostly preliminary results so far, but making progress (+ more infrastructure in place / lessons learned!)
- Lots of opportunities to use neural networks (and ML more broadly)
- But! Simple direct online optimization + simple model-based approaches in many cases may be more appropriate
- Much more interest from the accelerator community in the last couple of years
- Lots of potential for fruitful collaborations!

Thanks for your attention!

And many thanks to others who contributed to this work!

Sandra Biedron, Daniel Bowring, Brian Chase, Dave Douglas, Jonathan Edelen, Dean Edstrom Jr., Denise Finstrom, Dennis Nicklaus, Jinhao Ruan, James Santucci, Jim Steimel, Chris Tennant, and many others

Also, much of this work relied on Fermilab's HPC resources (thanks to Amitoj Singh, Alexei Strelchenko, Gerard Bernabeau, and Jim Simone!) and CSU's Summit system

Recap of Application Areas and Examples

- Model Predictive Control with Neural Network Models
  - Especially useful for systems with long-term time dependencies
    - *PIP-II RFQ*
    - *FAST RF gun*
- Modeling using Measured and/or Simulated Data
  - Create a fast simulation tool for online modeling
    - *FAST linac (later talk)*
    - *FEL energy switching study*
  - Create models from measured data alone
    - *JLab trajectory control*
    - *PIP-II RFQ*
    - *FAST RF gun*
  - Combine observed behavior and a priori knowledge
    - *FAST linac (later talk), PIP-II RFQ*
- Neural Network Control Policies
  - Tuning and changing operating state
    - *JLab FEL trajectory control*
    - *FEL energy switching study (see tomorrow's talk)*
  - Learning from existing control policies
    - *Present PIP-II RFQ work*
- Incorporating Image-based Diagnostics Directly into Control Policies
    - *FAST linac study (later talk)*
- Virtual Diagnostics
  - Predict beam parameters when diagnostic not available or not in use
    - *FAST linac study (later talk)*