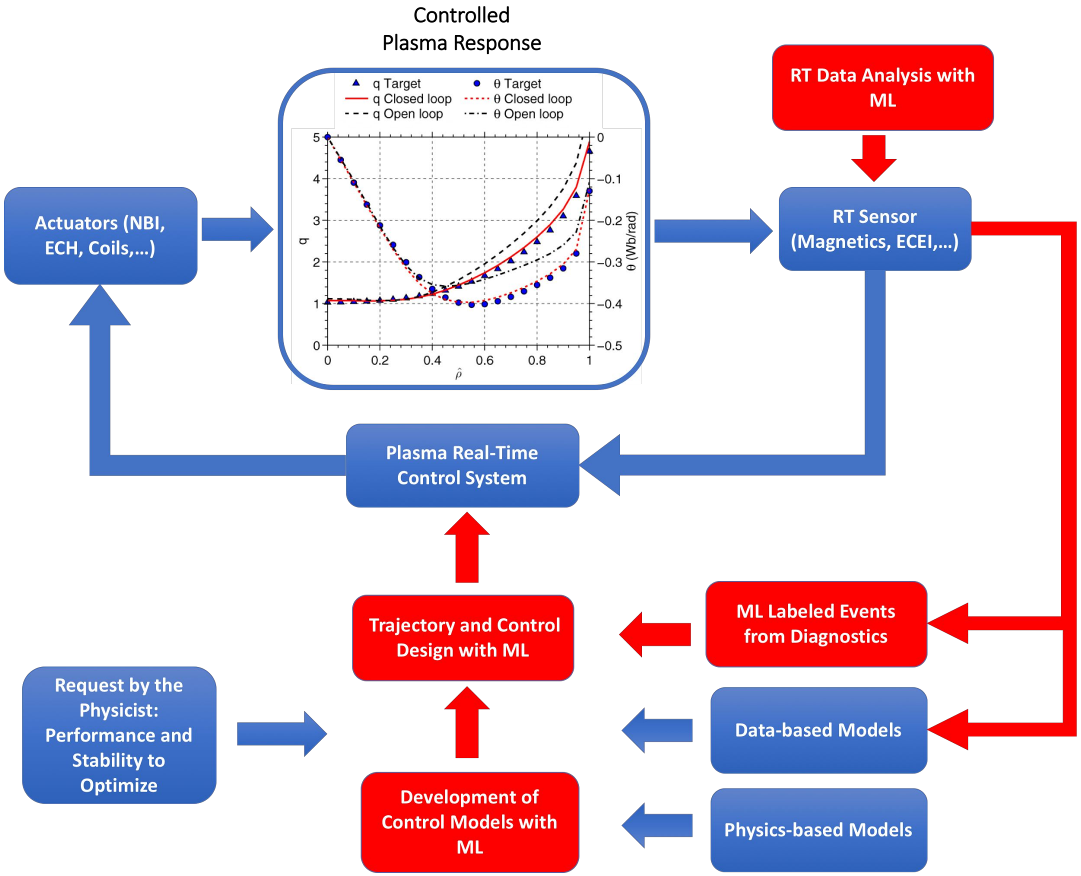
Machine Learning Predictions of Plasma Transport for Analysis and Control at DIII-D

Joe Abbate, Rory Conlin, Yichen Fu, Ricardo Shousha, Anthony Xing
PI: Egemen Kolemen

Magnetic Fusion Science Meeting
March 9, 2020

Birds-Eye View: Machine Learning for Plasma Control



- We would like to achieve a stable, high performance fusion reaction
  - Need real-time control + offline planning to obtain the states we want
  - Using both machine learning and reduced physics models
- Control actuators and actions are **mostly in the MHD to Transport timescales**

This Talk

- Traditional transport modeling
- Machine learning approach
- Results
- Applications to real time control
- Integrating machine learning into plasma control systems
- Future improvements

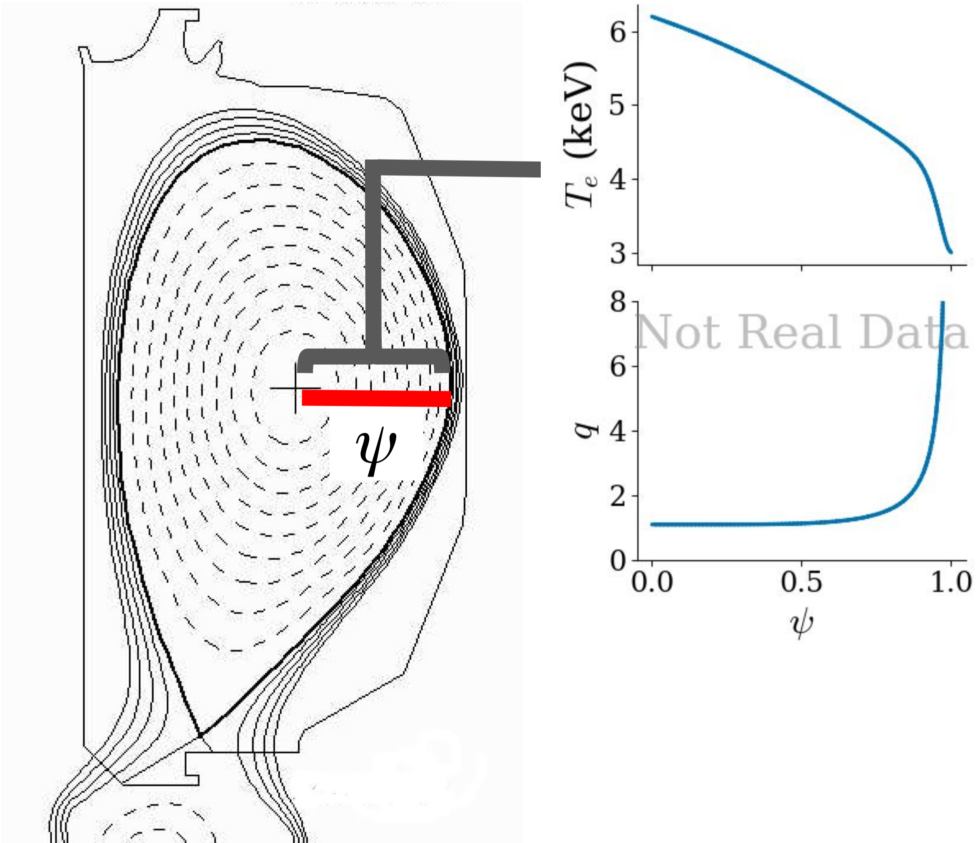
Goal:

- Predict plasma state evolution on transport timescales, given present state and actuator settings

Applications:

- Quick estimate of response to actuators
- Real time model-predictive control

Transport Plasma State



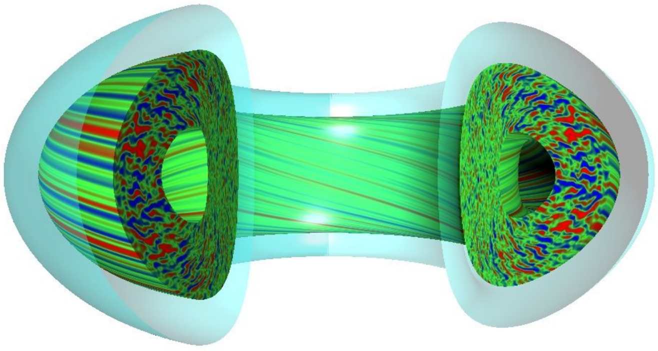
Full state of plasma determined by 1D profiles:

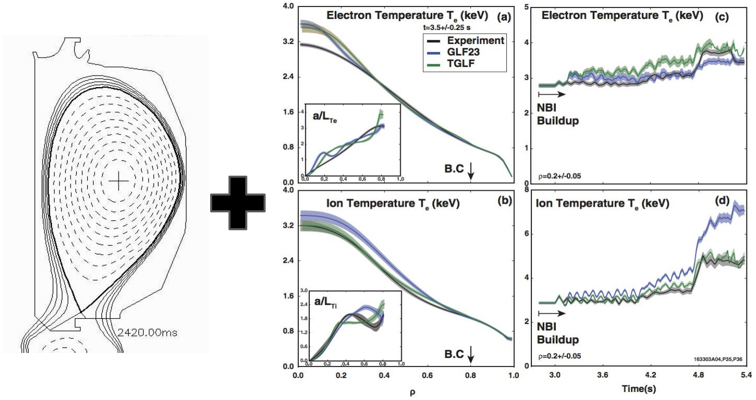
- Pressure (P)
- Current (J)
- Electron temperature and density (T_e, n_e)
- Ion temperature and density (T_i, n_i)
- Rotation (Ω)

Given state (and actuators), can we predict how plasma will evolve?

Traditional transport modeling

- Use gyrokinetic code to calculate transport coefficients
  - Full calculation (e.g. GYRO, XGC) takes ~months of CPU time
  - Quasilinear approximation (e.g. TGLF, QuaLiKiz) brings down to ~seconds-minutes
- Use transport equations to propagate profiles
  - e.g. TRANSP, ASTRA facilitate this
  - Requires power deposition (GENRAY for ECH, NUBEAM for NBI, etc)
  - Requires evolution of equilibrium (e.g. ISolver)





GYRO simulation from Greg Hammett's webpage
Meneghini et al 2018 Fusion Science and Tech 74

Downsides of traditional transport modeling

- Some approximations to future data often needed for convergence
  - Requires pedestal model
  - Need to fix some profiles to predict others
- Too slow for realtime use
  - Neural net approximations for transport coefficients, power deposition, pedestal models, etc. can possibly fix this (e.g. COTSIM, RAPTOR being developed)[1,2,3]
- Requires expertise and ad hoc setups
  - OMFIT is ameliorating this[4]
- Still discrepancies between experiments and simulations in some regimes[5]

[1] Meneghini et al 2017 Nucl. Fusion 57
[2] F. Felici et al 2018 Nucl. Fusion 58 (RAPTOR)
[3] E. Schuster et al 2019 Integrated Robust Control of Individual Scalar Variables in Tokamaks (COTSIM)
[4] Grierson et al 2018 Fusion Science and Technology, 74(1–2), 101–115
[5] S Smith et al 2015 Nucl Fusion 55

Fully Data-Driven Approach

Strengths

- **Fast**
- **Simple**
  - **No external models**
  - **No hand-tuning necessary**
- **Possibly more accurate in some regimes**

Weaknesses

- **Correlations only**
- **Only as strong as data**
- **Non-transferrable?**

ML successful in event prediction:

- FRNN *(Kates-Harbeck, 2019)*
- DPP *(Rea, 2019)*
- MLDA *(Fu, 2020)*
- TCN *(Churchill, 2019)*

- ~10,000 DIII-D shots from 2010-2018
- Train neural net on ~200k time samples

Learn f to predict state $\mathbf{\Delta t}$=200ms into future

$x(t)$:

- T_e
- n_e
- q
- Ω
- P
- *Shape*

$u(t)$:

- $P_{injected}$
- $T_{injected}$
- I_p
- $\langle n_e \rangle_{target}$

$$x(t+\Delta t) = x(t) + f[x(t), u(t)]$$

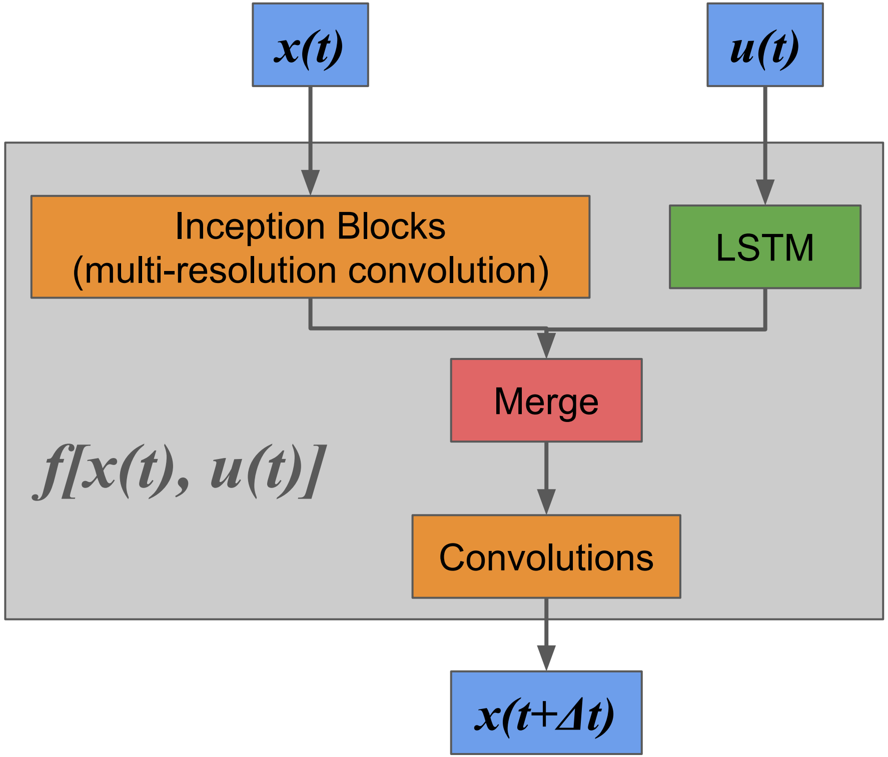
$x(t+\Delta t)$:

- T_e
- n_e
- q
- Ω
- P

Neural net to predict state **200ms** into future

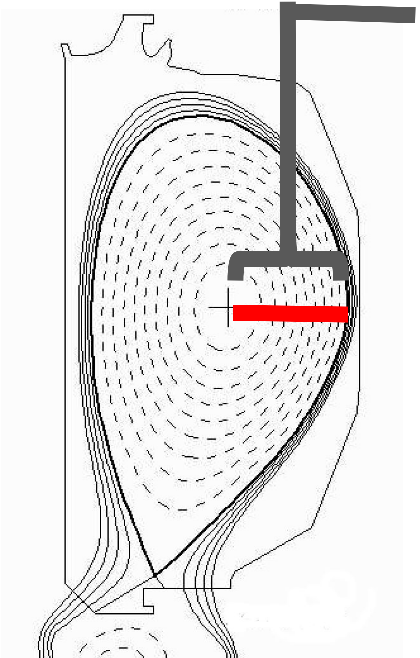
Model Architecture

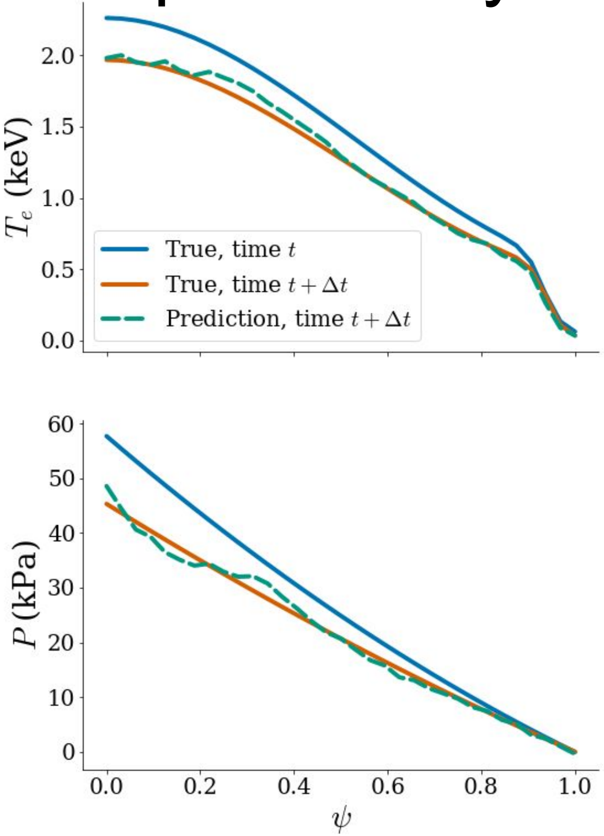
- Convolutional layers to capture gradients of profiles for transport (cf. natural response) *(Szegedy, 2015)*
- Recurrent layers to capture time history of actuators (cf. forced response) *(Gers, 1999)*



(Abbate,Conlin, in preparation)

ML predictions qualitatively accurate



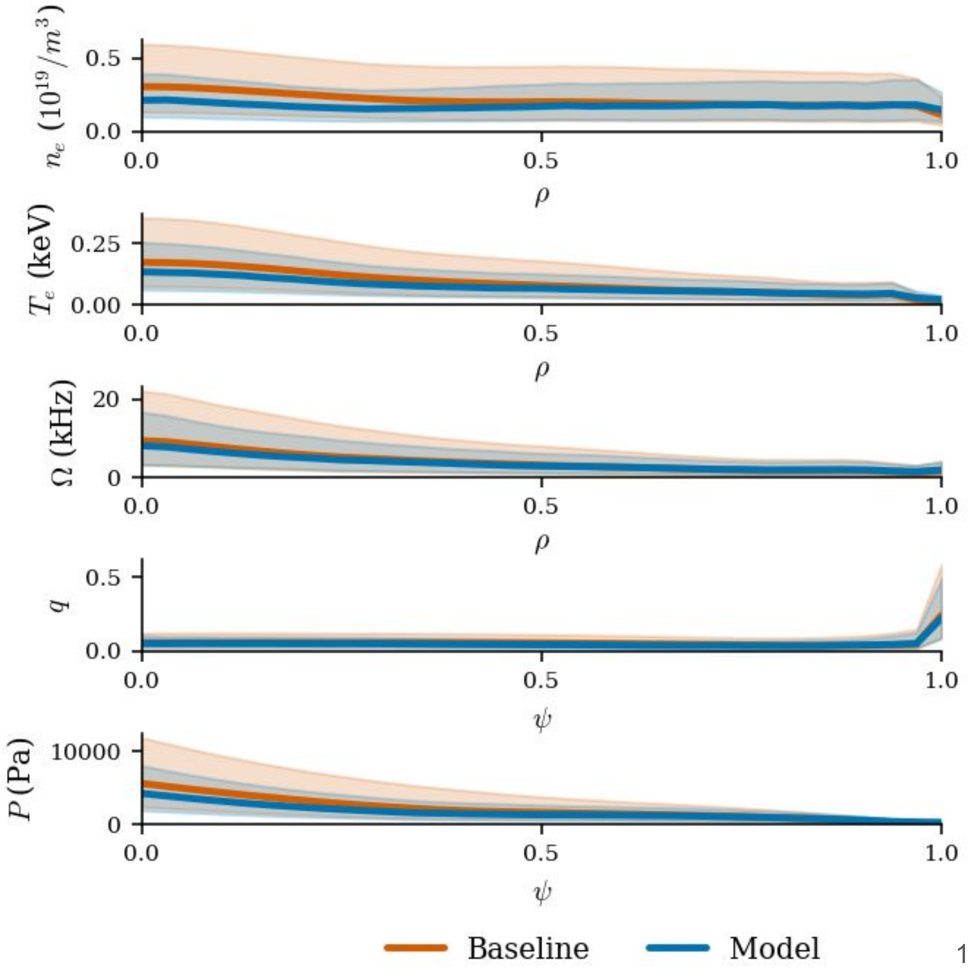


(Abbate, Conlin, in preparation)

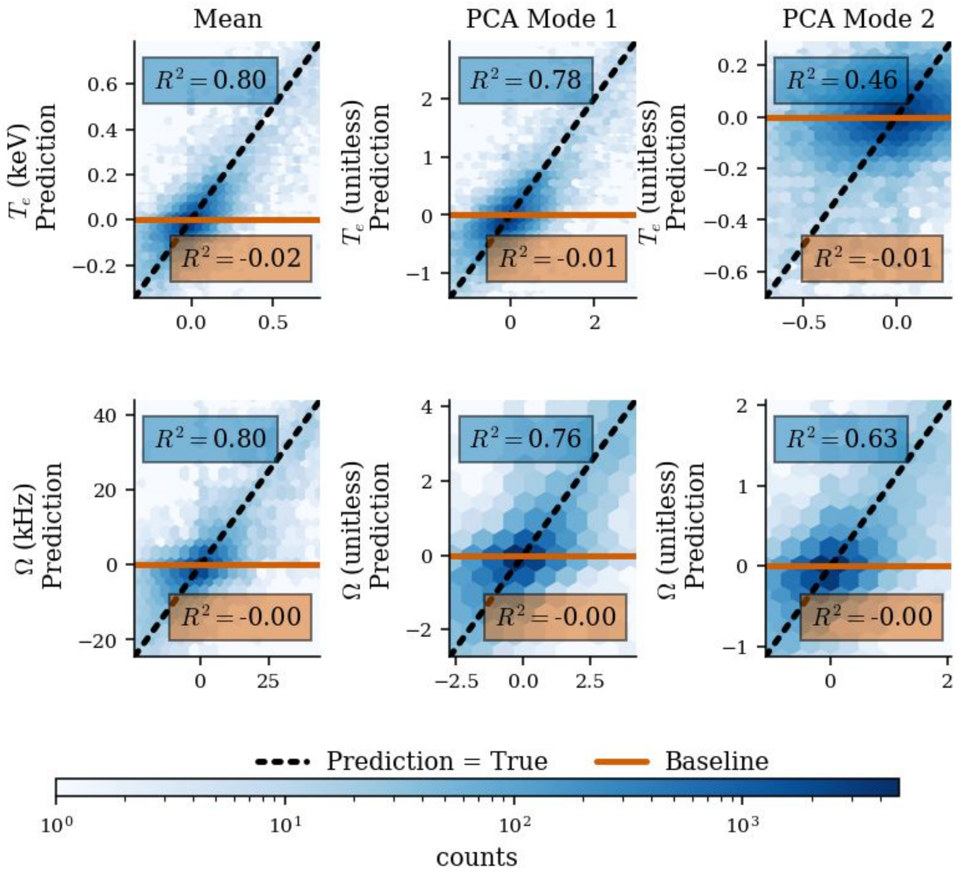
- Test on data from 2019 campaign
- Predicting 200 ms ahead
- Predictions noisy, but qualitatively correct

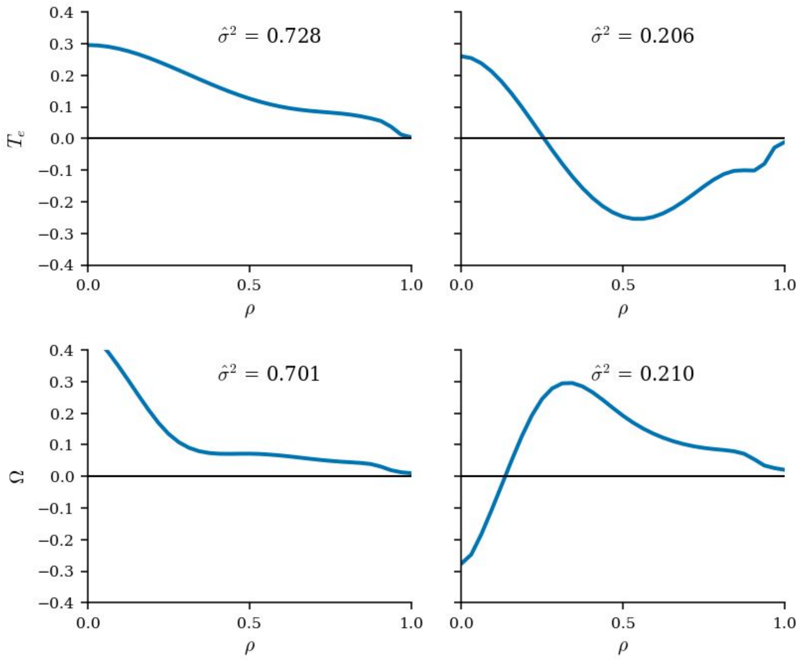
Preliminary results: ML median error better than baseline

- Baseline: predict no change in profile
- **Right**: Median absolute error over test set for baseline (orange) and ML predictions (blue)
  - **Lower is better**
- ML outperforms baseline prediction for all profiles



ML prediction captures most dominant modes



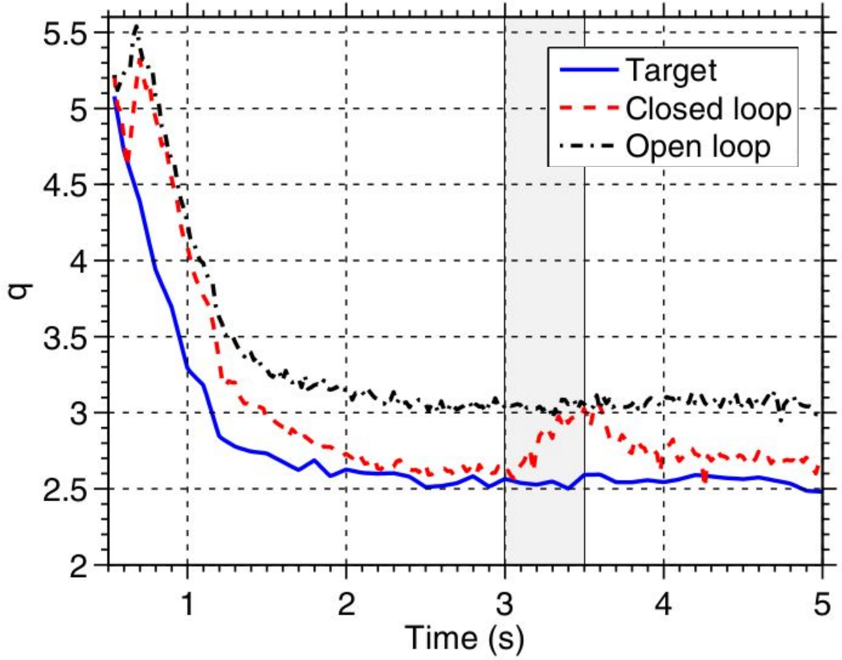


(Abbate, Conlin, in preparation)

Towards Profile Control

- Physics based control generally limited to controlling scalar variables
- Recent advances in profile control very limited

- Plasma is very complex, physics models for control are still lacking
- What can we learn from data?



(Boyer, 2013)

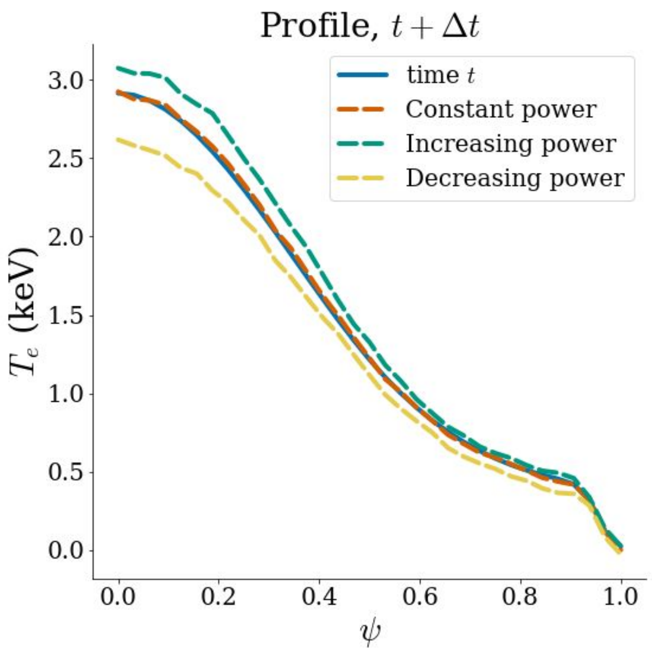
Model Predictive Profile Control

- Real time predictions allows predictive control
- Simulate different actions in real time

- Take the action to minimize cost function:

$$\mathbf{u}^*(t) = \underset{\mathbf{u}(t)\in U}{\operatorname{argmin}} \left\| \mathbf{w} \cdot \left[\mathbf{x}^{targ}(t+\Delta t) - \mathbf{x}^{pred}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \right] \right\|^2$$

$\mathbf{u}$: control action
$\mathbf{x}$: state
$\mathbf{w}$: weights



(Conlin, Abbate, unpublished)

How to integrate ML into real time control system?

- Neural Net developed in Python using Keras/Tensorflow
- Control system in C
- **Need a way to make neural net predictions from C**
- Existing approaches:
  - Call Python process:
    - ❌ Large latency, not safe for real time operations
  - Tensorflow C API:
    - ❌ Extremely difficult to code/implement
    - ❌ Relies on large external library
  - Tensorflow Lite:
    - ✔️ Supports limited operations in C for embedded systems
    - ❌ Doesn't support many required operations (recurrent networks, temporal convolution etc)

Developed Keras2c to run NN in real time

Script/Library for converting Keras neural nets to C functions

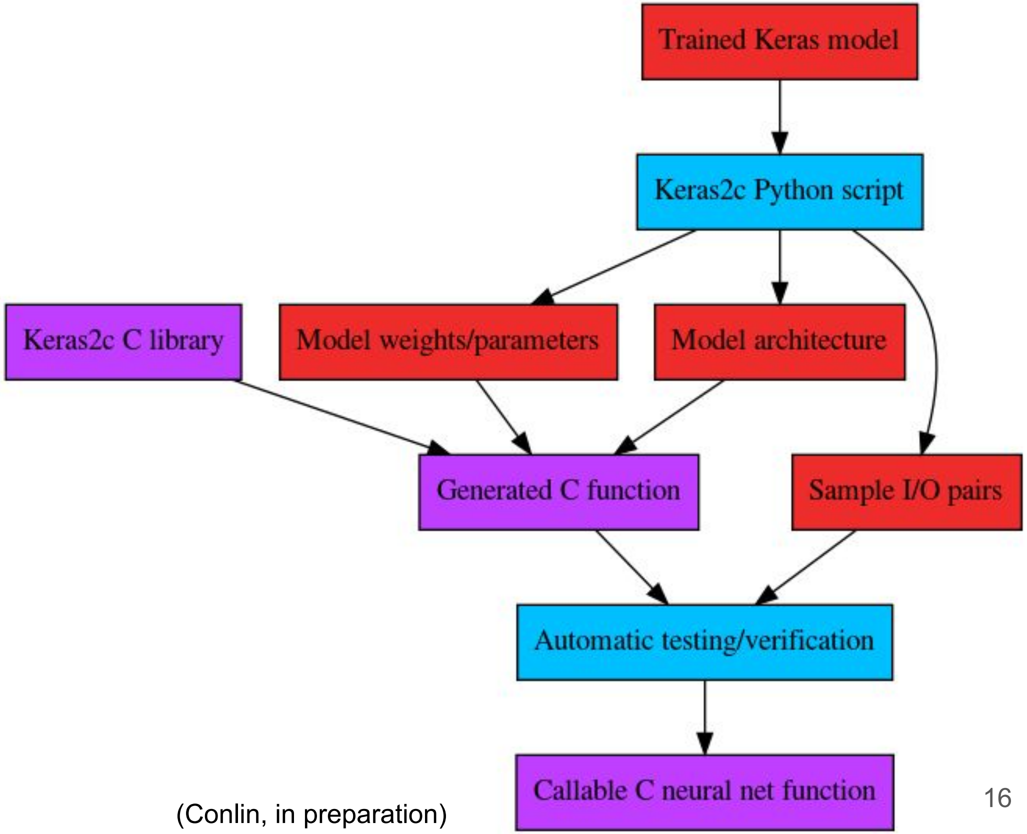
✓ Designed for simplicity and real time applications

✓ Core functionality only ~1300 lines

✓ Generates self-contained C function, no external dependencies

✓ Supports full range of operations & architectures

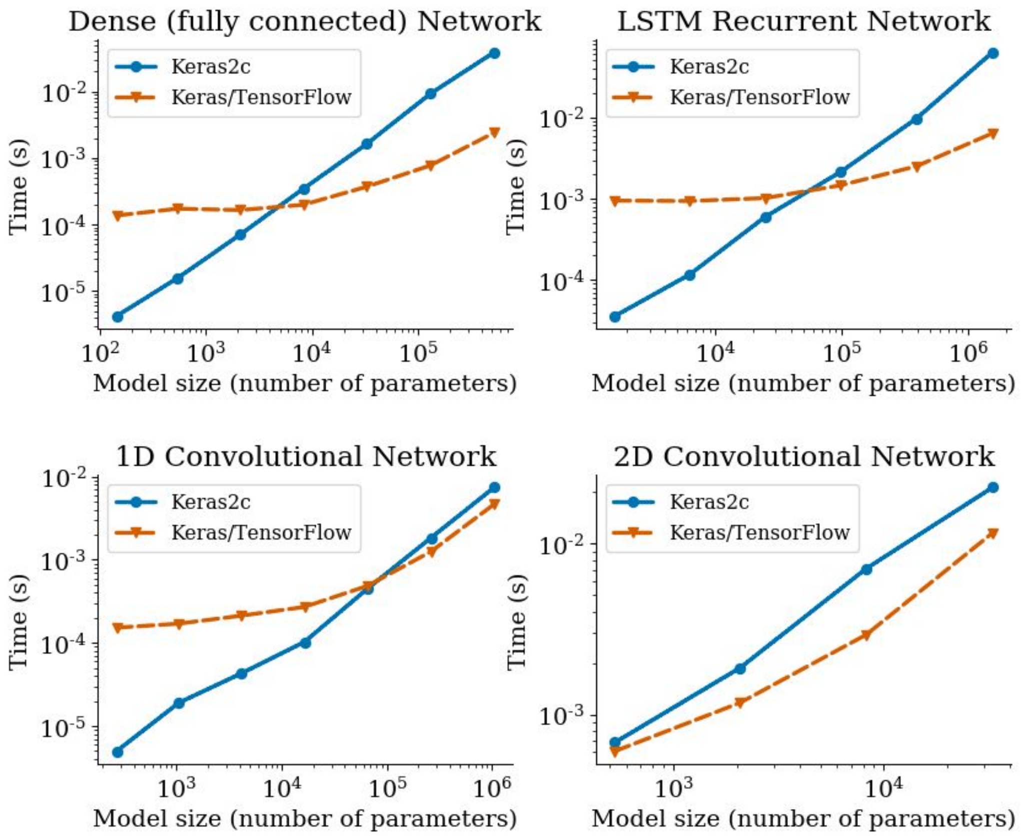
✓ Fully automated conversion & testing



(Conlin, in preparation)

Keras2c calculation speed comparable to Keras/Tensorflow

- Calculation time significantly faster than TensorFlow for small models
- Profile predictor model takes only ~600 μs per prediction
- Currently in use on DIII-D control system for profile prediction/control and disruption prediction (our group)
- Also, allows other groups to convert their NN to PCS code
- E.g. DIII-D FRNN (Bill Tang)

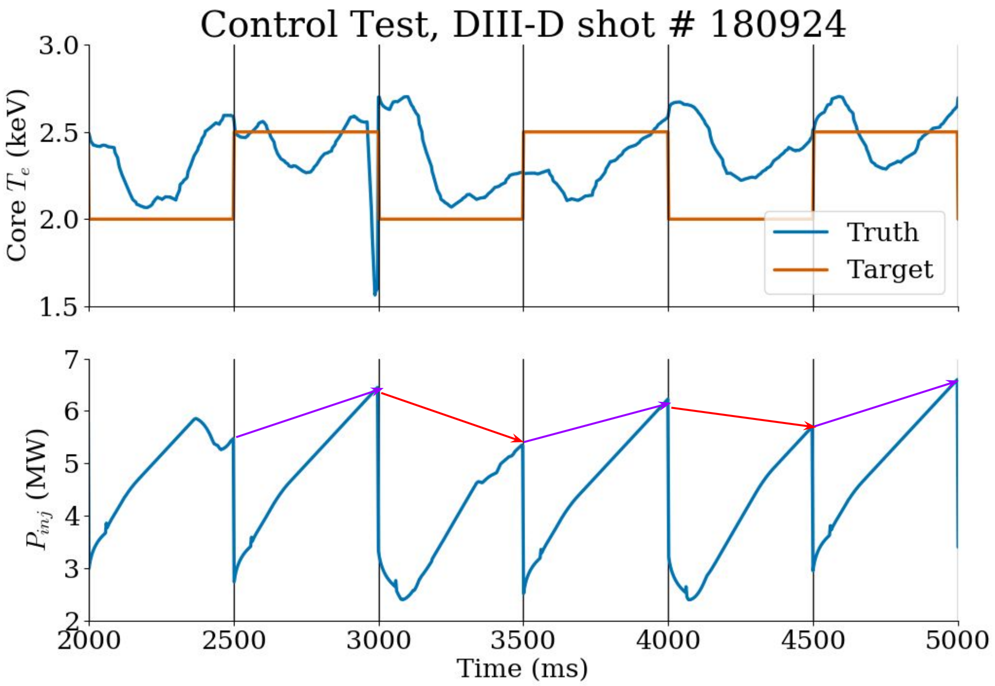


(Conlin, in preparation)

Profile Control: Preliminary Results

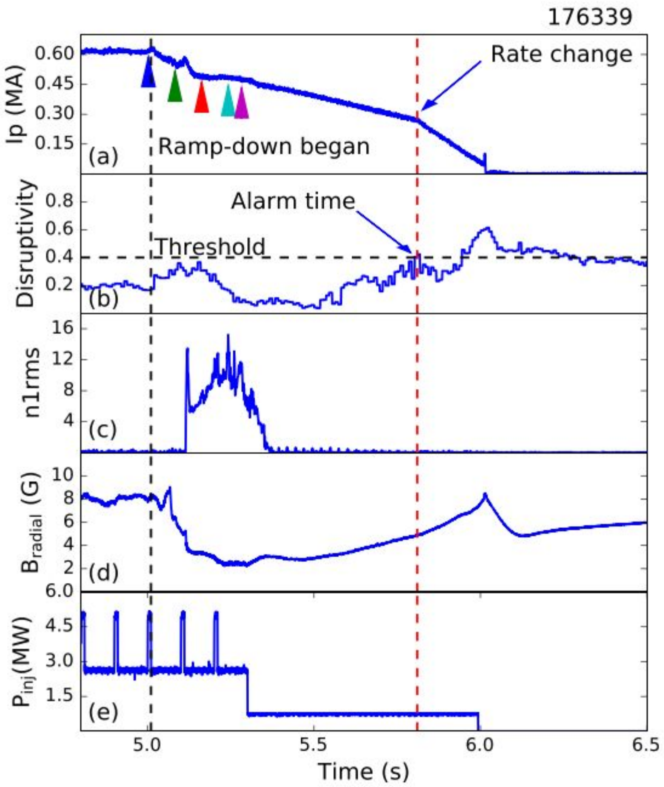
- Tested on 3 hour experiment on DIII-D, Nov 2019
- Alternating step target for core temperature
- Controller selected between 3 options for change in injected power
  - $\Delta P \in \{-150\text{ kW}, +0\text{ kW}, +150\text{ kW}\}$
  - $P_t = P_{t-1} + \Delta P$

- Bug in code caused model to lose "memory" whenever target changed
- Was still able to keep temperature close to target



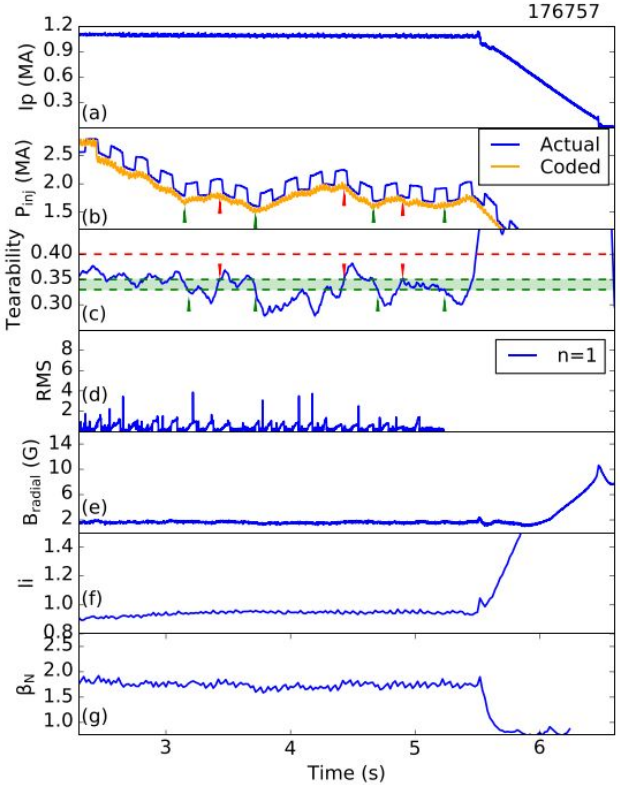
(Conlin, Abbate, unpublished)

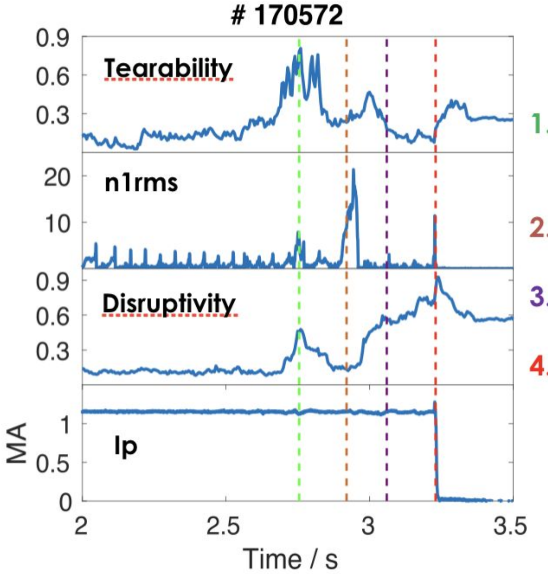
Work by Y. Fu on NTM/disruption prediction/control



- [Fu et al (2020) Physics of Plasmas]
  - featured article/Scilight,
  - featured in DOE press release
- ML algorithm to predict tearing modes & disruptions
- Gives ~250ms warning time
- Used to control rampdown to avoid disruptions

Next Step: Use “instability” in cost function for control



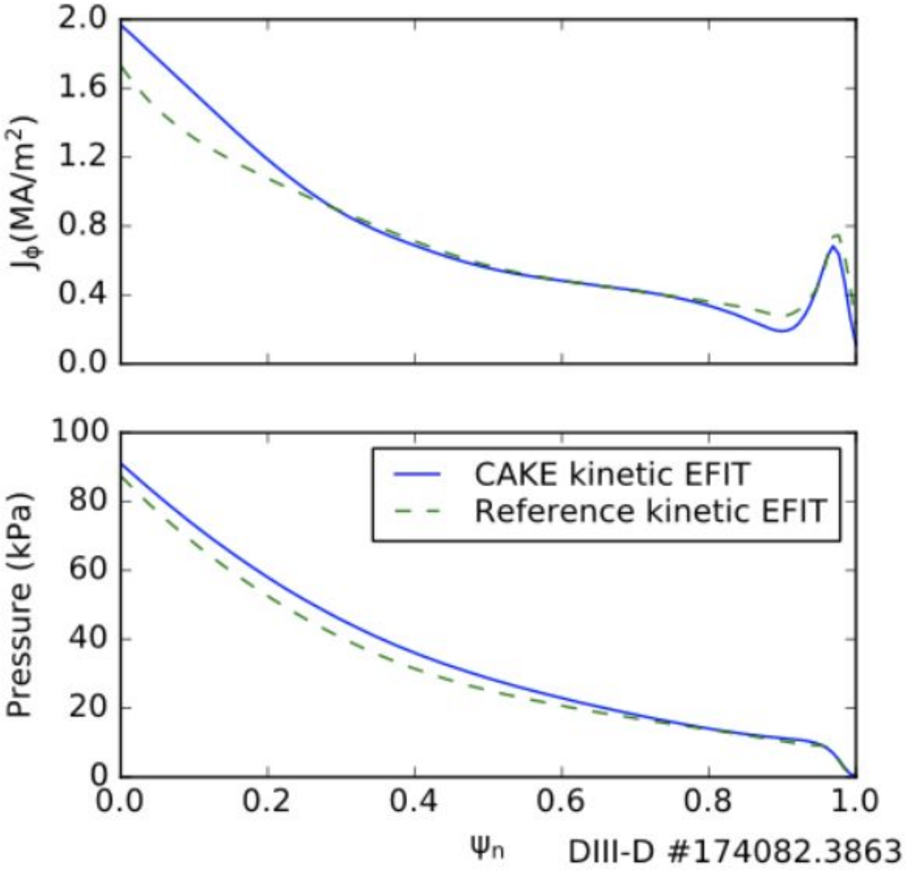


1. Predict instability (tearing mode) with ML
2. Instability (tearing) occurs
3. Predict disruption with ML
4. Disruption occurs

- Control to avoid disruption/NTM
- Simultaneously try to maximize plasma performance

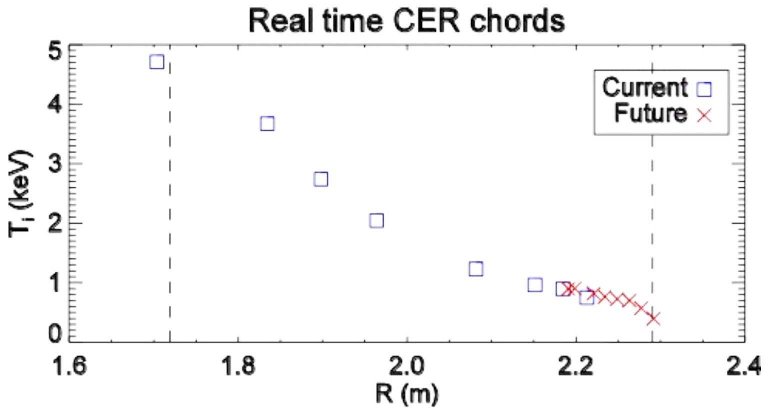
Next Step: Higher Quality Profile Database with CAKE

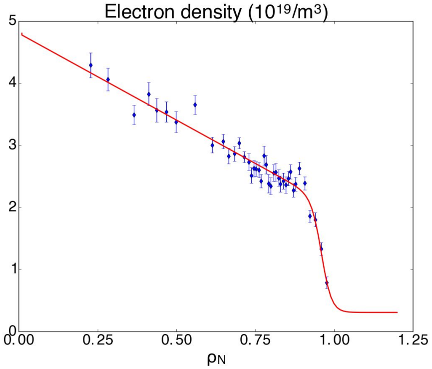
- Currently ML model is trained on EFIT01 + Zipfit
  - Low quality fits, no pressure pedestal, no MSE constraint
- Consistent Automatic Kinetic Equilibria (**CAKE**) code produces kinetically constrained reconstructions with little to no human tuning
- Compares well with manually fit kinetic equilibrium

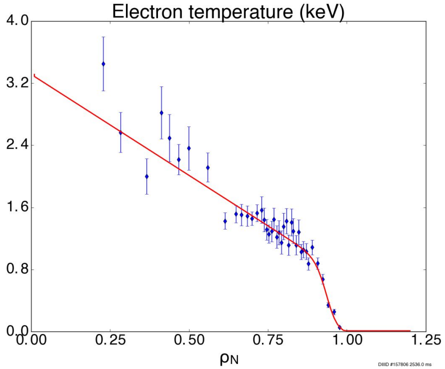


Xing et al (2020) submitted to PPCF

Improved Real time diagnostics for RT CAKE





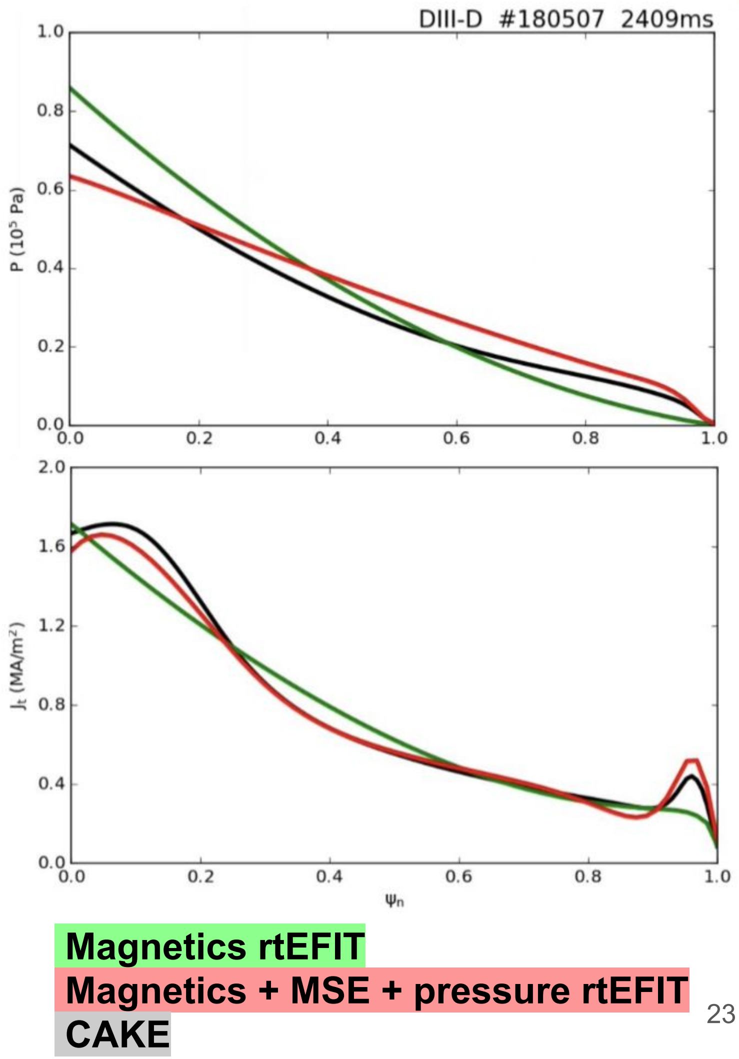


Group's Contribution RT-Diagnostics:

- Real-time Thomson analysis added 2015.
- RT-Edge CER and analysis added 2018/9
- RT-ECE system (used for Alfven Mode Control)
- NN Fast ion calculation (FIAT)
- Real time measurements of T_e, n_e, T_i, Ω
- Allow better predictions from ML models in real time

Next Step: Real time CAKE

- Estimate pressure profile using Thomson scattering / CER
- Use estimated pressure profile + MSE as additional constraint in rtEFIT
- Pressure profile results:
  - Correct pedestal
  - General agreement, but stiffer than CAKE
- Current density profile results:
  - Correct bootstrap peak
  - General agreement with CAKE
- Improved state estimates = improved ML predictions



R. Shousha (in preparation)

Summary

- Profile predictor works well for offline analysis
- Predictive control tested on DIII-D
  - Demonstrated effective temperature control via NB power
  - Further tests to control more profiles
  - Integrate disruption/NTM avoidance using profile control
- Keras2C allows easy integration of ML models into PCS
  - Used for profile control, disruption prediction (FRNN group) on DIII-D
- CAKE / real time CAKE will provide reliable kinetically constrained equilibria & profiles for both ML training and real time control

Backup slides

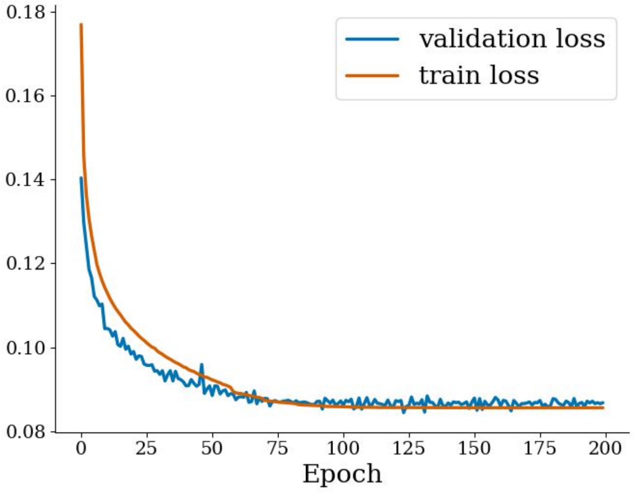
Training set criteria

DIII-D shots from 2010 through the 2019 campaign are collected from the MDS+ database. Shots with a pulse length less than 2s, a normalized beta less than 1, or a non-standard topology are excluded from the start. A variety of non-standard data is also excluded, including the following situations:

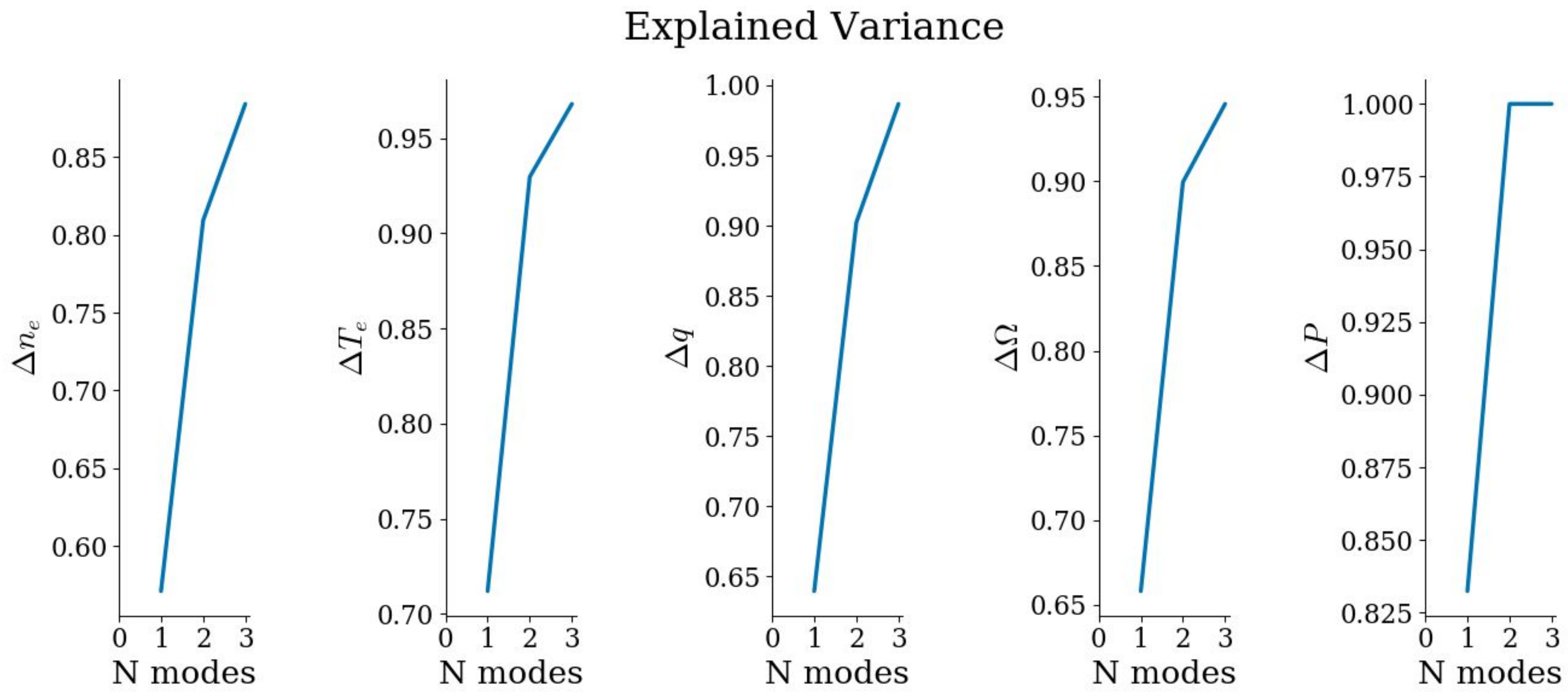
- during and after a dudtrip trigger
- during and after ECH activation, since ECH is not currently included as an actuator
- whenever density feedback is off
- during and after non-normal operation of internal coils
- for shots where any needed signals are not in the database

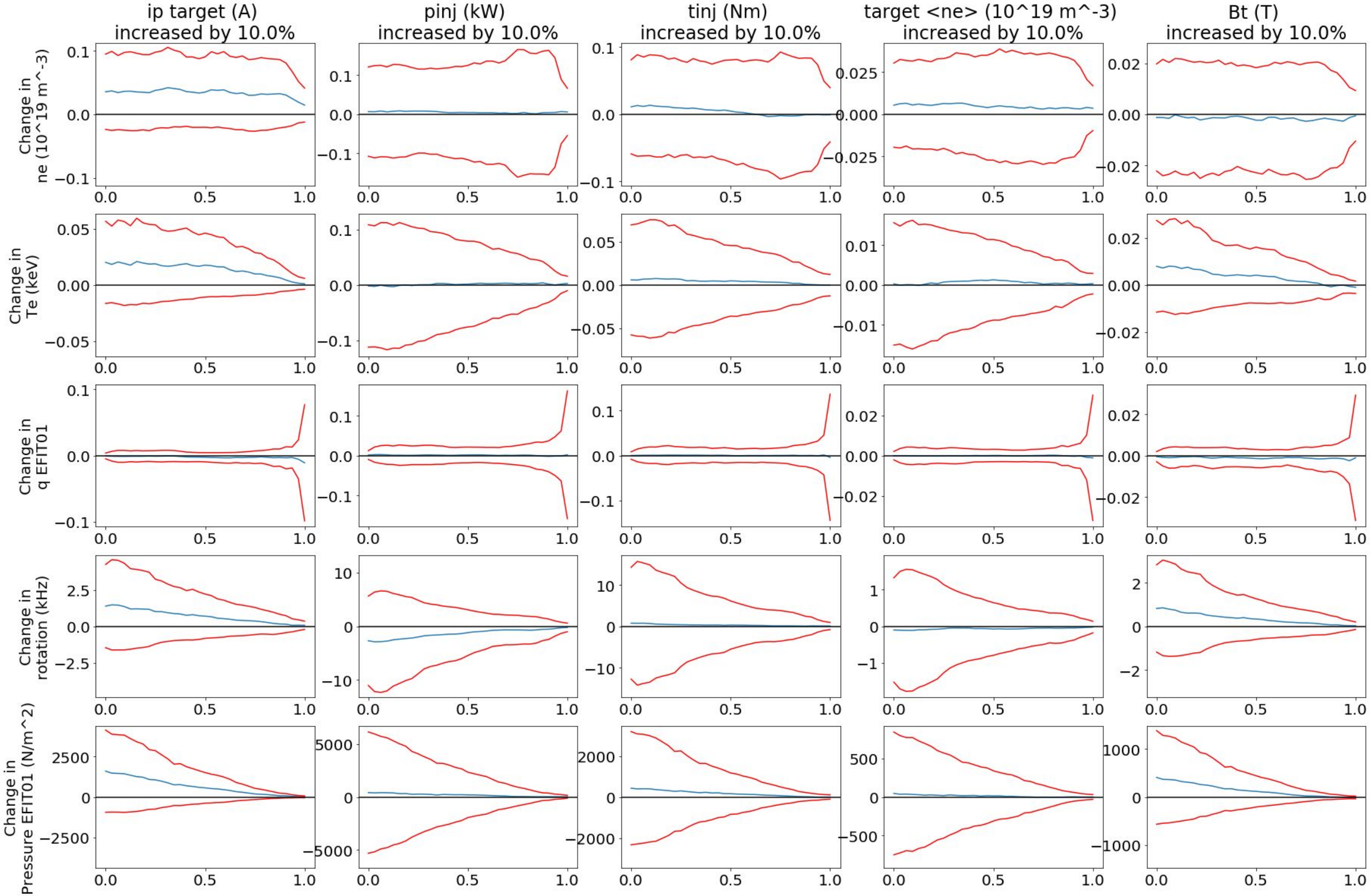
Neural Net Info

- Lookback: 6x 50ms windows
- Lookahead: 200 ms
- Normalization: subtract median, divide by IQR
- Data source: Zipfit, EFIT
- Loss function: weighted MSE
- Optimizer: Adagrad
- Batch size: 128
- Epochs: 200
- Trainable parameters: 166,887
- Inception block size: 2x16, 4x16, 8x16
- Activation: ReLU
- Input signals:
  - Profiles: Electron temperature, electron density, q, rotation, pressure
  - Scalars: line averaged density, inductance, minor radius, divertor separation, triangularity, plasma volume
  - Actuators: injected power, injected torque, plasma current, target density, toroidal magnetic field
- Outputs: Electron temperature, electron density, q, rotation, pressure

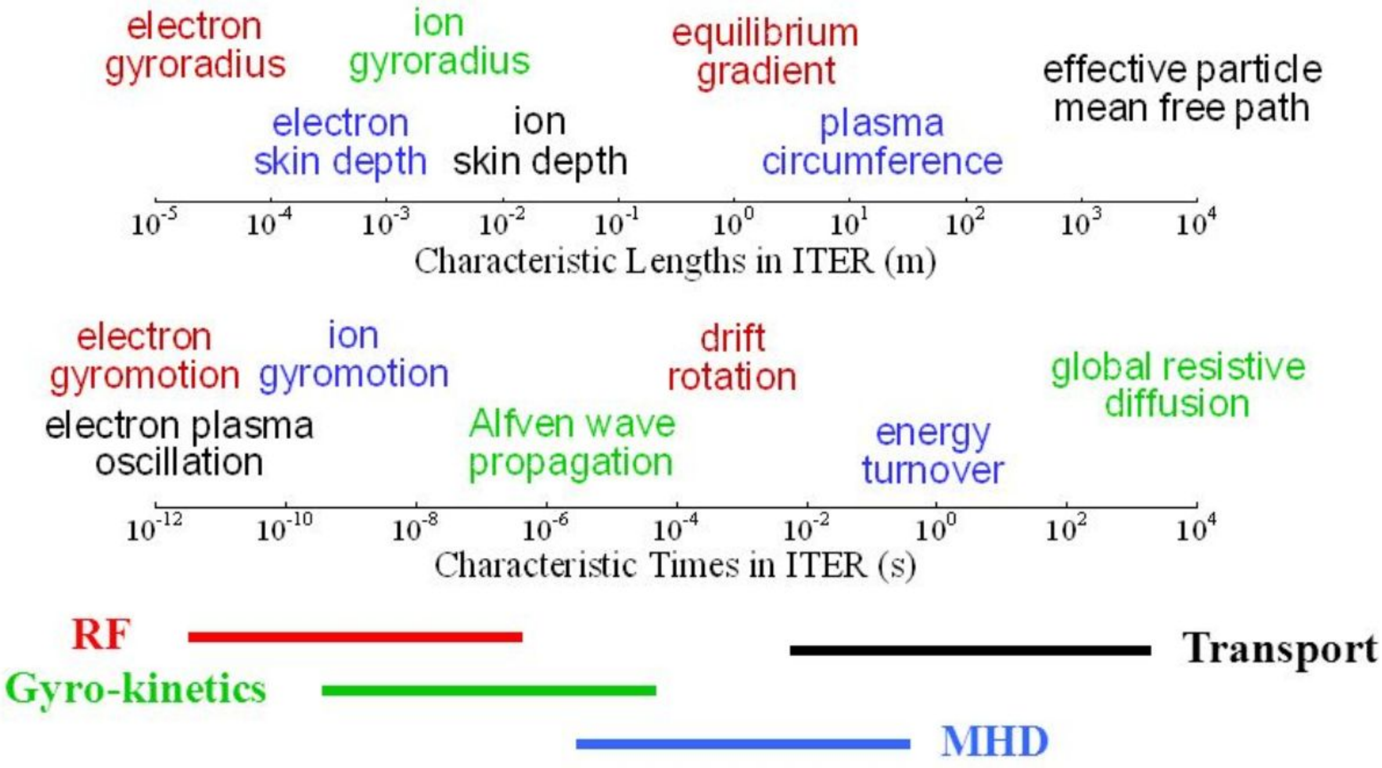


PCA explained variance

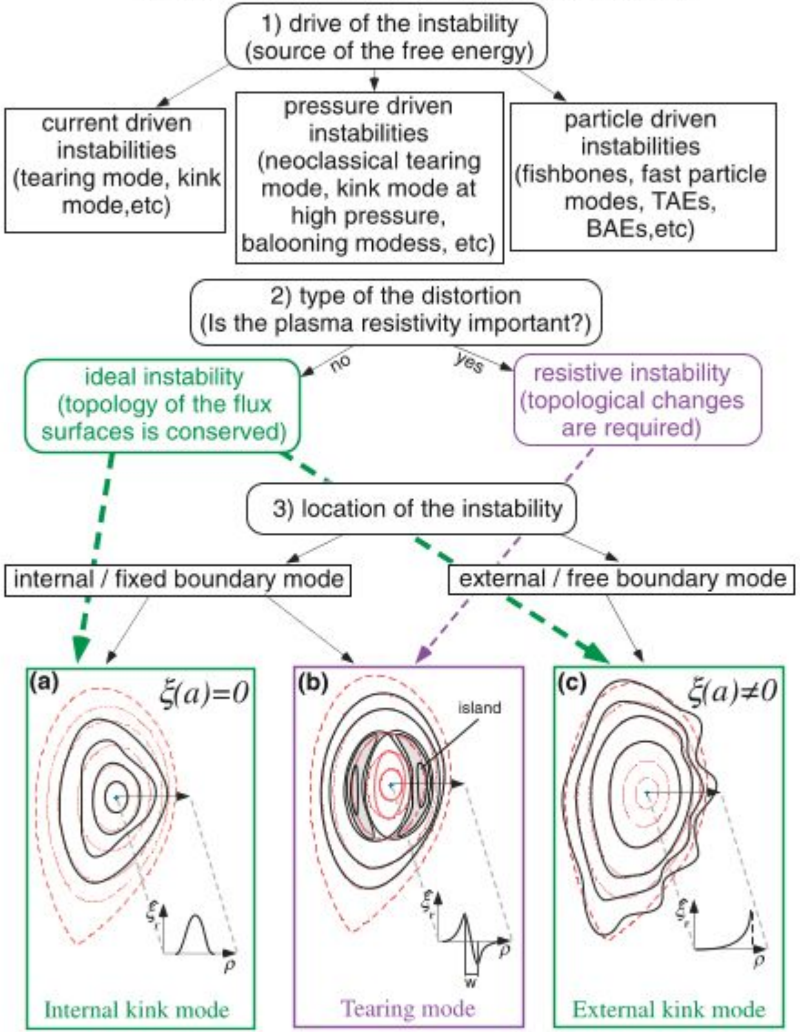



ip target (A) increased by 10.0%
pinj (kW) increased by 10.0%
tinj (Nm) increased by 10.0%
target <ne> (10^19 m^-3) increased by 10.0%
Bt (T) increased by 10.0%
Change in ne (10^19 m^-3)
Change in Te (keV)
Change in q EFIT01
Change in rotation (kHz)
Change in Pressure EFIT01 (N/m^2)

Plasma Timescales



Basic classification of MHD instabilities



Transport Equations

With sources of n_e, $L_\mathrm{t} \equiv \rho_m \langle R^2 \rangle \Omega_\mathrm{t}$ and p_s, transport equations are

$$\text{density} \qquad \frac{1}{V'}\frac{\partial}{\partial t}\bigg|_{\psi_\mathrm{p}} n_e V' + \dot{\rho}_{\psi_\mathrm{p}} \frac{\partial n_e}{\partial \rho} + \frac{1}{V'}\frac{\partial}{\partial \rho}(V' \Gamma) = \langle \overline{S}_n \rangle,$$

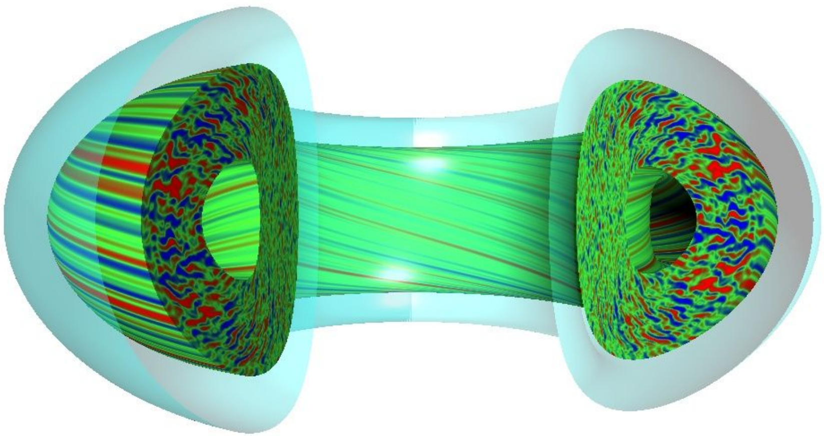
$$\text{tor. mom.} \qquad \frac{1}{V'}\frac{\partial}{\partial t}\bigg|_{\psi_\mathrm{p}} L_\mathrm{t} V' + \dot{\rho}_{\psi_\mathrm{p}} \frac{\partial L_\mathrm{t}}{\partial \rho} + \frac{1}{V'}\frac{\partial}{\partial \rho}(V' \overline{\Pi}_{\rho\zeta}) = \langle \vec{e}_\zeta \cdot \Big(\overline{\vec{J} \times \vec{B}} - \vec{\nabla} \cdot \overline{\overleftrightarrow{\Pi}} + \textstyle\sum_s \overline{\vec{S}}_{\mathrm{p}s} \Big) \rangle,$$

$$\text{energy} \qquad \frac{3}{2} p_s \frac{\partial}{\partial t}\bigg|_{\psi_\mathrm{p}} \ln p_s V'^{5/3} + \frac{3}{2} \dot{\rho}_{\psi_\mathrm{p}} \frac{\partial p_s}{\partial \rho} + \frac{1}{V'}\frac{\partial}{\partial \rho}(V' \Upsilon_s) + \langle \vec{\nabla} \cdot \vec{q}^{\,\mathrm{pc}}_{s*} \rangle = \overline{Q}_{s\mathrm{net}}.$$

J.D. Callen 2014 Lectures on Tokamak Plasma Transport Modeling

Gyrokinetic codes

- 5-dimensional
- First-principles
- ~months of CPU time
- Examples: GYRO, XGC
- Reduced physics models:
  - ~10s of seconds to a day
  - TGLF
  - QuaLiKiz

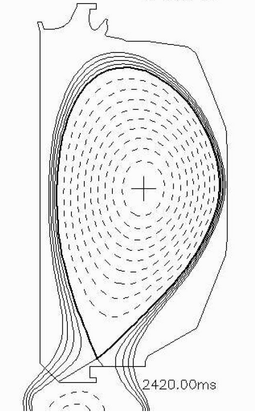


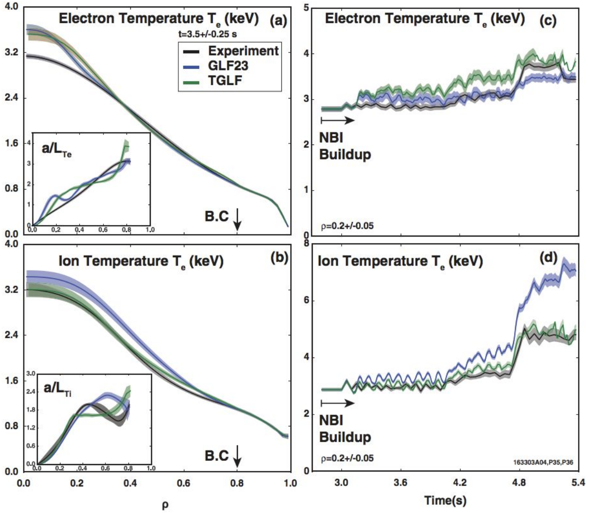
Get transport coefficients

Picture: GYRO simulation from Greg Hammett’s webpage

Predictive transport codes

- 1.5 dimensional
- Additional codes
  - Power deposition
  - Pedestal models
- Solves the transport equations
- Examples: TRANSP, ASTRA





Update profiles

[1]Ongena et al 2012 *Numerical Transport Codes*
[2]Meneghini et al 2018 Fusion Science and Tech 74 (origin of picture)