WPJET1-PR(18) 21145

J Kates-Hardbeck et al.

Accelerating progress towards controlled fusion power via deep learning at the largest scale

Preprint of Paper to be submitted for publication in Nature

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Progress towards controlled fusion power via deep learning at the largest scale

Julian Kates-Harbeck,[1,2*] Alexey Svyatkovskiy,[3] William Tang[2,3]

[1]Department of Physics, Harvard University,
17 Oxford St, Cambridge, MA 02138, USA
[2]Princeton Plasma Physics Laboratory, Princeton, NJ, USA
[3]Princeton University, Princeton, NJ, USA

[*]To whom correspondence should be addressed; E-mail: jkatesharbeck@g.harvard.edu.

Nuclear fusion power via magnetic confinement tokamak reactors carries the promise of sustainable and clean energy production for the future [1]. The avoidance of large-scale plasma instabilities called disruptions [2, 3] is one of the most pressing challenges [4, 5] towards this goal. Disruptions are particularly deleterious for large reactors such as the multi-billion dollar international ITER project [6] currently under construction, whose reaction for the first time aims to produce more power from fusion than is injected to heat the plasma. Here, we present a new approach to forecast disruptions based on deep learning that for the first time overcomes the crucial limitations of past work, such as first-principles based [5] and classical machine learning approaches [7, 8, 9, 10, 11]. In particular, our method for the first time (i) delivers reliable predictions on machines other than the one it was trained on — a crucial requirement for large future reactors that cannot afford "training"

disruptions; (ii) utilizes high-dimensional training data such as profiles to add new physics information and boost predictive performance; and (iii) engages supercomputing at the largest scale to deliver solutions with improved accuracy and speed. Trained on experimental data from the largest tokamaks in the US (DIII-D [12]) and the world (JET [13]), our method moreover can be tuned for physics-specific tasks such as prediction with long warning times, and opens up possible avenues for moving forward the goal from passive disruption prediction to active reactor control and optimization. These results illustrate the potential for deep learning to accelerate progress in fusion energy science and in the understanding and prediction of complex systems in the physical sciences in general.

Tokamaks use strong magnetic fields to confine high-temperature plasmas with the goal of creating the conditions for extracting power from the resulting fusion reaction in the plasma [14]. The thermal and magnetic energy in the tokamak can drive plasma instabilities that lead to disruptions [2], a central science and engineering challenge facing practical power production from nuclear fusion. Disruptions abruptly destroy the plasma's magnetic confinement, thus terminating the fusion reaction and rapidly depositing the plasma energy into the confining vessel [3, 4, 15]. The resulting thermal and electromagnetic force loads can irreparably damage key device components. However, if an impending disruption is predicted with sufficient warning time [3], a disruption mitigation system (DMS) using techniques such as massive gas or shattered pellet injection [16] can be triggered. The DMS terminates the discharge but significantly reduces the deleterious effects of the disruption. Current guidance for the minimum required warning time for successful disruption mitigation on ITER is about 30 ms, although it is in general set by the exact response time of the DMS and may be reduced in the future through progress in DMS technologies [3]. Throughout this paper, we describe the predictive performance of all methods at this “deadline” of 30 ms before the disruption.

While plasma instabilities and disruptions are in theory predictable from first principles [17], this has proven to be extremely challenging since an accurate physical model [5] would need to take into account (i) a vast range of spatio-temporal scales; (ii) multi-physics considerations; and (iii) the complexity of disruption causes and precursor events [18]. Just like for many other fundamental questions across the physical sciences [19, 20], the inherent complexity of the problem can make first-principles based approaches impractical on their own.

On the other hand, recent statistical and “classical” [21] machine learning (ML) approaches [7, 8, 9, 10] based on real time measured data have shown promising results. While they still have several shortcomings, these pioneering efforts represent the current state of the art [3] for disruption prediction. Here we introduce FRNN (Fusion Recurrent Neural Network), a new disruption prediction method based on deep learning (DL) that overcomes their key limitations. Specifically, our method for the first time (i) utilizes the information contained in high-dimensional diagnostic data such as profiles in addition to using scalar signals; (ii) avoids the need for extensive feature engineering and selection [22, 23]; (iii) enables rapid training times by engaging high performance computing; and (iv) delivers predictive capabilities on devices unseen during training. This cross-device prediction in particular will be key for future powerful machines like ITER, since they cannot be run to disrupt more than a few [3] times. Accordingly, training data from such devices can be expected to be scarce.

Deep neural networks [24] in general consist of many layers of parametrized nonlinear mappings, whose parameters are trained (“learned”) using backpropagation. They have been successful at learning to extract meaningful features from high-dimensional data such as speech, text, and video. In particular, recurrent neural networks (RNNs) powerfully handle sequential data by maintaining information in an internal state that is passed between successive time steps, in addition to taking into account new input data at every time step. Meanwhile, convolutional neural networks (CNNs) can learn salient, low-dimensional representations from high-dimensional data by successively applying convolutional and down-sampling operations. As the first application of deep learning to disruption prediction, the specific architecture of FRNN combines both recurrent and convolutional components to extract spatio-temporal patterns from multi-modal and high-dimensional sensory inputs. The overall workflow and detailed architecture of our approach are presented in figure 1.

Missing a real disruption or calling it too late (false negative, FN) is costly because its damaging effects go unmitigated, while triggering a false alarm (false positive, FP) wastes experimental time and resources. Changing the alarm threshold value for the scalar "disruptivity" output of a prediction model (see figure 1 (d)) allows a trade-off between these two economic factors. A low threshold means the alarm is triggered more easily, which will result in fewer fewer missed disruptions but more false alarms, and vice versa for a high threshold. This trade-off is captured as a receiver-operator characteristic (ROC) curve [25] (see extended data figure 1 as well as methods for details). The area under this ROC curve (AUC) — our metric for evaluating algorithms in this paper — lies between 0 and 1 and measures the ability of a predictive method to catch real disruptions early enough, while at the same time causing few false positives.

In order to asses our algorithm, we train it to predict disruptive and non-disruptive outcomes on experimental data from the Joint European Torus (JET) and DIII-D tokamaks, currently comprising over 2 TB. Training our model effectively required solutions to several unique challenges such as training with diverse and long sequences and finding machine agnostic signal normalizations (see methods for details, along with a detailed summary of the data and signals used). We compare FRNN to the previous state of the art of support vector machines (SVMs) [10] and small multi-layer perceptrons (MLPs) [8], as well as other promising models from the ML literature such as random forests [26] and gradient boosted trees [27]. Table 1 reports AUC values for the best version of our model and the best classical model on various data sets. In all our tests, gradient boosted trees performed the best among classical models.

On the DIII-D data set, we sample both the training and testing examples uniformly across all experimental runs ("shots"). Thus, this data set requires the least "generalization" — i.e. the ability of the algorithm to learn patterns during training that transfer to new and possibly unseen situations, in this case the testing set. Here, classical methods and our proposed method are competitive, with the classical method performing slightly better. However, FRNN improves further in performance closer than 30 ms to the disruption (providing improved predictive performance if mitigation technology becomes faster in the future), performs as well as the classical method in the "interesting" [3] region of the ROC curve with high TPs and low FPs, and provides better generalization for threshold choices (see extended data figure 1).

In the JET data set, training and testing data are drawn from slightly different distributions. The testing set is from after an upgrade to the device where the internal wall was changed from a carbon wall (CW) to an "ITER-like wall" (ILW) made of beryllium [13], resulting in different physical boundary conditions as well as different shot and operations characteristics [10]. Here the superior generalization abilities of FRNN become clear.

Being able to learn generalizable disruption-relevant features from one tokamak and apply them to another will be key for a disruption predictor for ITER, where no extensive disruption campaigns can be executed for generating training data. The second and third column of table 1 show the results for cross-machine performance, where both training and validation data come from one machine, and testing is performed on the other. This is a difficult task, complicated by various subtle factors (see methods), which has proven challenging for earlier work [11]. The results show that in this setting, only our DL approach is able to transfer significant generalizable knowledge from one machine to the other. The results are particularly strong for the ITER relevant case of training on the machine with smaller physical size and less stored energy (DIII-

D) and generalizing to the “big” unseen machine (JET). As far as we are aware, this is the first demonstration of significant cross-machine generalization for ML based disruption prediction.

While it is not possible to obtain thousands of training shots (including a sufficient number of disruptions) from a new machine like ITER, a small amount of simulated or real (perhaps low power/current) disruptive shots [3] may be feasible. To simulate this scenario, we sample a small set δ of shots from the testing set on the “big” machine (JET) and give the algorithms access to these during training (see methods). Encouragingly, all models significantly benefit from this “glimpse” at the testing set (see last column in table 1). Generalization is particularly strong for the DL model. Using only very few JET shots, FRNN is able to reach performance competitive with that of models trained on the full JET data set on the same restricted set of signals available on both machines. These results are highly relevant for disruption prediction on ITER, since they demonstrate the feasibility of training well-performing models without the need for many disruptive training shots from the target machine.

Since manual dimensionality reduction and feature engineering (i.e. the extraction of useful low dimensional summaries or representations from high dimensional data [28]) would first be necessary, classical methods have been unable to take advantage of higher-dimensional signals such as profiles. Profiles are 1D data capturing the dependence of a relevant plasma parameter such as the electron temperature or density on the radius as measured from the plasma core to the edge. This radial dependence is generally the most important, as variations along the poloidal or toroidal degrees of freedom are subject to much greater particle mobility and resulting faster averaging times due to the structure of the confining magnetic fields [14]. Such profiles for example could provide rich (disruption-relevant) physics information and insight, and many reaction metrics and control mechanisms already relate to their temporal evolution [15, 29]. While profile data show significant differences between machines and currently are of limited quality and temporal availability (see methods), our algorithm is nonetheless able to extract significant information from these profiles and generalize between machines. Performance of the best models universally increases when including profiles (see table 1), including for cross-machine prediction. This demonstrates that there is a wealth of disruption-relevant information contained in multi-modal, high-dimensional data — a critical fusion physics insight. These findings are further corroborated by explicit analyses of signal importance (see extended data figure 2). The new ability of our DL model to take advantage of this data without resorting to the use of hand-tuned features or invoking human domain expertise is key. Higher-quality, more densely available, and potentially even higher-dimensional signals such as 2D ECEi imaging data [30] (see methods for additional examples) will add even more predictive power to DL models and might lead to new physics insights in the future.

Figure 2 shows time series of various example shots and the resulting algorithmic predictions. In (a), an example false alarm is triggered on a DIII-D shot at about 5200 ms into the shot. However, this FP remains a plausible prediction, as the observed symptoms are consistent with a “minor disruption” — an event characterized by a thermal quench (i.e. a rapid loss of thermal energy to the surrounding vessel) without a current quench (i.e. a loss in plasma current) [3, 15]. Accompanied by only minor disturbances in the plasma current, this is evident in the (i) drop in β (the ratio of thermal to magnetic pressure in the plasma); (ii) peaking and rapid change of the temperature (and density) profiles; and (iii) spiking locked mode.

The fact that false alarms are often understandable like this and “make sense” gives confidence and physical interpretability to the model. By serving as a reliable measure of “disruptivity” as exemplified in this shot, FRNN could serve as an analysis tool to help identify causes, precursors and other events relevant to disruption physics [2, 31], thus supporting discovery science in this area [15] — a useful side effect of FPs. As is visible from the random spikes in (b), we find qualitatively that false alarms from the classical methods are often erratic and not as attributable to physically meaningful events.

Figure 2 (b) shows an example of a disruptive shot that is missed by the best performing classical algorithm (gradient boosted trees) but is correctly caught by our method. Although no sudden events occur near the disruption at the end of the shot, FRNN does pick up on the slow (~ 1000 ms long) rise of the core radiated power, while the best performing classical approach — gradient boosted trees — does not. This is likely due to its lack of access to temporal information (see methods).

Figure 2 (c) compares FRNN trained (and tested) on only scalar signals (yellow) with a model trained on all signals including profiles (black) on a disruptive DIII-D shot. As can be seen from the drop in β, the morphological change in the profiles and the locked mode spikes (as well as the later spikes in radiated power), starting around 3350 ms some events are clearly taking place in the plasma that resulted in a disruption. However, only the model trained using profiles is able to correctly interpret the early warning signs. Access to 1D profile information qualitatively changes the prediction and allows early detection of the disruption, which is missed by the model without access to profiles.

Optimizing a modern ML model is an iterative process. Selection of well-performing hyperparameters — i.e. parameters of the model that are not optimized during training and need to be set manually, such as the learning rate — requires searching a high-dimensional space (see extended data table 3 for a comprehensive list of hyperparameters and values that were found to perform well). Evaluating any point in this space entails running full model training and inference. To make this approach practical, it is key to reduce the time required to train a single model, and to increase the amount of models that can be trained in a given time (see methods for details). Growing model sizes, data sets, and amounts of 1D or even higher-dimensional data will only make these demands more challenging.

We address these issues with three levels of parallelism, which together enable engagement of high performance computing (HPC) at the largest scale to reduce the time to solution. First, GPU (graphical processing unit) computing accelerates training over single-machine, multi-core CPU execution by $\sim 10-20$x. Using the Message Passing Interface standard (MPI), we next implement a distributed, synchronous, data-parallel training approach [32] to engage large numbers of GPUs at once. Finally, we parallelize the random hyperparameter search by training many such distributed multi-GPU models in parallel.

An important application of hyperparameter tuning is the ability to tune models for a specific physics task — such as providing much earlier disruption warnings, thus possibly enabling active plasma control without the need for shutdown [3]. In figure 3 (a) we show the results of using hyperparameter tuning to select models for optimal prediction performance at 30 and 1000 ms before the disruption, respectively. The tuned models display qualitatively distinct behavior which generalizes to the testing set: the model tuned for 30 ms shows better performance closer to the disruptions, while the model tuned at 1000 ms shows superior performance at times further away.

Figure 3 (b) demonstrates the excellent strong scaling of FRNN's data parallel training up to at least 6000 GPUs on the OLCF Titan supercomputer. We have replicated this scaling on the Pascal-P100-powered TSUBAME 3.0 and Volta-powered OLCF Summit supercomputers, as well as with mixed floating point compute precisions [33]. In the inset to figure 3 (b), we study training progress as a function of wall time multiplied by the number of GPUs. The fact that the curves approximately collapse indicates that actually training a model to convergence also scales nearly ideally with the number of GPUs employed.

Figure 3 (c) shows the results of hyperparameter tuning runs on 100 parallel random models, each trained with 100 GPUs, engaging a total of 10^4 GPUs. We compare performance to scenarios of only engaging 1 or 100 GPUs to perform the same search. The black curve shows the speedup of using 10^4 GPUs over single GPU execution as a function of the AUC of the models found. Parallel search becomes increasingly effective for higher AUC values, since those values occur more rarely. The inset shows near perfect speedup in finding the *best* model using 10^2 and also 10^4 GPUs, indeed demonstrating effective engagement of supercomputing systems of $O(10^4)$ GPUs — the scale of the largest supercomputers available today [34] — and a resulting overall time to solution of only half an hour.

Ultimately, the goal will be not just to mitigate disruptions but to avoid them entirely if possible. Models that learn a salient representation of the state of the reactor — as the method presented here — could lie at the core of a deep reinforcement learning [35] approach. Using training reactors or simulated data with synthetic diagnostics [36], these models could be trained to directly control the reactor while minimizing disruptivity and also optimizing arbitrary objectives such as fusion power output. This also highlights the potential for future synergy between ML and more traditional modeling and simulation efforts.

With the example of disruption prediction in fusion reactors, this paper demonstrates that DL can complement the other pillars of discovery science — theory, simulation and experiment — in the analysis, prediction, and control of a highly complex physical system. With the rapidly growing availability of multi-modal and high-dimensional data across disciplines, these findings — as well as some of the specific associated challenges and insights — have implications for the applicability of DL to the physical sciences in general.

Methods

Data considerations

Data and preprocessing The data for individual experimental runs (or "shots") are stored as separate time traces for every signal, with sampling periods between $\sim 1 \times 10^{-5}$ and $\sim 1 \times 10^{-1}$ seconds. For each shot, we read in all relevant signals, and cut the signals to the range of times during which all signals contain data. We then resample the signals to a common sampling rate of 1 millisecond (ms) using causal information (i.e. for any given time always using the last known value before the time in question).

Each timestep contains a vector of n signals (see extended data table 1). For multidimensional signals, their values are simply concatenated onto the global input vector. A single shot then contains $n \times T$ scalar values where T is the length of the shot. The full data set includes several thousand shots from both the Joint European Torus (JET) and DIII-D tokamaks. Only shots that have data for all signals are included. See extended data table 2 for a summary of the full data set. Overall, the size of our data set from DIII-D and JET amounts to about 2 TB [37].

Data challenges A fusion plasma is a complex dynamical system with an unknown internal state which evolves according to physical principles and emits a time series of observable data [14]. Capturing the history and current physical state of the plasma should allow predictions about its future behavior, including the possibility of disruption. Noisy and incomplete data make this a challenging statistical task.

Observable data are captured as scalars and 1D profiles by various passive diagnostics such as magnetic measurements, electrical probes, visible and UV spectroscopy, bolometry, electron cyclotron emission (ECE), X ray measurements; as well as active diagnostics such as Thomson scattering, LIDAR, interferometry, or diagnostic neutral beams [38]. Future work may also consider higher dimensional sources of data such as such as 2D ECEi imaging [30], 2D magnetic equilibria [39], or fast camera data [40].

Raw experimental data is difficult to work with directly using machine learning methods. For instance, the relevant physical time scales and experimental sampling frequencies of the different signals span several orders of magnitude. While many dynamic variables in the plasma change within ms or faster, each experimental run (or "shot") can last anywhere from $\sim 1 - 40$ seconds. We choose a time step of 1 ms to resolve the fastest relevant dynamics without including excessive data. This sampling results in training examples with sequence lengths of order $O(10^4)$.

For each shot, if there is a disruption, this only occurs at the end. Moreover, depending on the machine, disruptions can be quite rare ($< 10\%$ of shots). This means the actual learning signal for disruption events is quite sparse. We used up-weighting [41] of positive examples (see the hyperparameter λ in extended data table 3) to stabilize training and found that it often was able to increase performance.

Signals are often noisy or only exist partially. We only use shots that have at least some data for every desired signal. However, in contrast to past work we do not exclude any shots based on "bad" or statistically unusual data. Some signals in the experimental databases are computed using non-causal information (e.g. temporal averaging with a time-centered window, usually with a width of ~ 20 ms). We shift such signals in time to ensure the algorithm does not have access to any future information at any given time. This approach means that for some signals the algorithm is seeing slightly "old" data, giving a conservative estimate for prediction performance.

Some signals are not stored consistently in the database. For instance, the input power signal on DIII-D changed its units from MW to kW around shot 156000. This was not corrected for during our analysis. Since the algorithm divides all shots by the same numerical scale, shots before this change appear to the ML algorithms incorrectly to have a very low value of the input power. Thus, the signal importance of the input power on DIII-D is likely underestimated by extended data figure 2.

Profile data currently available are of limited quality and temporal resolution. Profiles are available at best every 20 ms for DIII-D and 50 ms for JET, and are often poorly reconstructed or missing entirely. They are also shifted in time to accommodate for non-causal filtering in the EFIT equilibrium reconstruction. Additionally, the data are qualitatively different between machines, consisting of noisy raw data on JET and smooth fitted functions on DIII-D.

The computer science community has established a strong example of unified, open data sets (e.g. ImageNet, IMDB, Penn Treebank, etc.) [42, 43, 44] against which new machine learning methods can be tested. This allows a direct and fair comparison between various methods and leads to measurable incremental progress. In practice, the separation and complexity of the various international experimental facilities make the construction of such unified databases more challenging for the fusion community. Thus, most data currently exists in separately managed databases. We have taken the approach of implementing not only our own method, but a generalized interface that allows a user to plug and play any machine learning algorithm adhering to a “train” and “predict” application programming interface (API). This allows direct comparison and benchmarking between variants of our RNN approach with other ML methods, including the state of the art as used in past publications, such as SVMs and MLPs, as well as recently popular classical methods such as Random Forests or Gradient Boosted Trees [26, 27]. We believe that such a direct comparison on the *same exact data* is key to accurately measure progress and to allow detailed comparisons of the relative strengths and weaknesses of certain methods. To simplify database access once permission has been obtained, we have included in the code base [45] object oriented code based on human readable signal names that fetches raw data from the appropriate original databases and performs error checking — this is key for generating training data sets reliably and at scale.

We have found empirically that absolute predictive performance can be quite sensitive to ad-hoc choices about the data set such as the precise group of shots that are used (and which ones are excluded due to bad/abnormal data, intentional disruptions, or other criteria) and which signals are used. In our approach, we use all shots from a given time period. We exclude shots only if for any of our desired signals the shot does not contain data at all. This means that our data set includes shots with known bad data, intentional disruptions, testing shots, etc. While this can hurt performance, it is the most conservative and least ad-hoc approach. Overall, improved handling of the data issues mentioned in this section may raise absolute performance beyond the levels reported in this paper. Thus, while absolute performance numbers are important and will be key for the application of disruption prediction to ITER, we also invite the reader to pay particular attention to the *relative* performance between different methods, as these highlight their relative strengths and weaknesses.

Algorithm and training details

Training the neural network effectively requires overcoming several unique challenges, such as the need for generalizable signal normalization, poorly defined target functions not directly related to the ultimate learning objective (high area under the ROC curve), and a need for stateful training [28] on very long ($O(10^4)$) sequences of varying length. In this section we describe our approach for overcoming these challenges in our training procedure. We also provide a comprehensive list of tunable hyperparameters for our model in extended data table 3. All deep learning (DL) models were implemented using Keras [46] and Tensorflow [47].

Normalization Neural networks typically expect their inputs to lie in similar numerical ranges across all dimensions. Moreover, they expect a signal of equal amplitude to have equal meaning across examples. This poses a significant challenge in the use of raw physical signals as inputs to any NN architecture. Since the raw signals have values in the range of 10^{-6} to 10^{19}, the signals must be normalized such that they all lie around 10^0. Moreover, many signals (such as the plasma current, stored energy, or even the time scale itself) will have *differing characteristic scales* on different tokamak machines. The normalization should ideally have the property that signals having the same "physical meaning" from different machines get mapped to the same numerical value after normalization. As suggested in past work [11], physically motivated dimensionless combinations of the raw measurements are a sensible option for generating such input data.

However, we find empirically — the particular normalization scheme used is in essence a tunable hyperparameter of the model just like any other — that the best performing method is to simply normalize each signal by its "global numerical scale" across the entire data set. This automatically brings signals to a reasonable numerical range and scales appropriately to different tokamak devices. Thus, the "normalized form" of each signal (which is how signals are plotted in figure 2 and how the actual algorithm receives them) is simply the original signal value divided by this global numerical scale, which is computed as follows. For each shot, we compute the standard deviation of a single signal across that shot [48]. Then we define the "global numerical scale" of that signal as the median across all shots of those per-shot standard deviations. Since a small fraction of shots contain strong outlier data points lying orders of magnitude outside of their typical range (which could distort the computation of the standard deviation), the median provides a resilient way of obtaining aggregate scale information from all shots. No shots are removed or filtered out from the data sets for having outlying or unusual data. To further ensure that outliers do not deteriorate performance, we also clip each signal to lie within $[-100\sigma, +100\sigma]$, where σ is its corresponding numerical scale, although we find that this does not measurably affect performance.

To make profiles scalable between machines, they are at every time step stored not as a function of real spatial position, but rather as a function of normalized toroidal magnetic flux (ρ). In extended data table 1 we give a comprehensive list of signals including their respective units and global numerical scales.

Target functions The ultimate goal of this learning project is to predict the onset of disruptions. The exact definition of what target function the neural network should learn to approximate is important for the architecture of the model and ultimately for its performance. While ultimately a shot is either disruptive or not (i.e. the decision is binary, 0 or 1), the RNN needs to return an output value at every time step. For a non-disruptive shot, the output should clearly always be 0 or "non-disruptive". However, in a disruptive shot, the best choice for the "target output" is less obvious. While shortly before the disruption the output should be 1 or "disruptive", this is not necessarily true several seconds before the disruption. It is also unclear which choice for such a target function would ultimately result in the highest possible area under the ROC curve (AUC) — the ultimate performance metric we are trying to optimize.

Our solution defines a parameter $T_{warning}$ such that the target function is 1 if the time to disruption is $T_D - t < T_{warning}$ and 0 otherwise ($T_D - t$ is the time to disruption, where T_D is the time at which the disruption occurs and t is the current time). The intuition is that the neural network *shouldn't be able to know* about a disruption more than $T_{warning}$ away. Setting $T_{warning}$ too high might lead to many false positives, while setting $T_{warning}$ too low might cause the algorithm to fail to learn "early warning signs" of disruptions. On JET for instance, we find empirically that values of $T_{warning} \sim 10$ s work best. We also tried predicting $T_D - t$ or $log_{10}(T_D - t)$ directly using a regression loss function. The log version performs well for the DIII-D tokamak, but not on JET.

We also implemented a "max hinge" loss in the hopes of more closely approximating the ultimate learning objective: a high ROC area. This loss merely considers the maximum output value across all time steps and penalizes it if it doesn't cross the threshold in a disruptive sample or if it does cross the threshold in a non-disruptive sample. The penalty is an $L1$ hinge loss with threshold -1 for non-disruptive time steps and threshold $+1$ for time steps within $T_{warning}$ of a disruption. The intuition is that in the final evaluation of a shot, only the maximum value of the network matters: either it triggers an alarm or not. Thus, this loss should give a more direct incentive for the network to optimize the area under the ROC curve. In practice, we find that "max hinge" performs about as well as a standard hinge loss with the same parameters (for the standard hinge loss, the same same loss is applied individually for every time step, not just at the time step of maximum output).

A user of a deployed version of this predictive system must define an *alarm threshold*, such that when the RNN output signal reaches a certain value, an alarm is triggered and thus disruption mitigation actions are engaged. This alarm threshold allows the user to trade off between maximizing *true positives* (TP) and minimizing *false positives* (FP). A true positive is a true disruption that is correctly caught by the algorithm (i.e. an alarm is triggered). A false positive is an alarm that is triggered even though there wasn't going to be a disruption. We define the TP rate as the fraction of real disruptive shots for which the algorithm triggers an alarm before the 30 ms deadline. The FP rate is the fraction of nondisruptive shots for which the algorithm triggers an alarm at any point in time. As the alarm threshold is raised (harder to cause alarms), there will be less false positives, but also less true positives. As the threshold is lowered (easier to cause alarms), there will be more false positives, but also more true positives. By varying the threshold, an ROC curve which plots the TP rate vs the FP rate (see extended data figure 1) is traced out that describes the predictive performance of the algorithm holistically. To capture this overall tradeoff, we use the area under this ROC curve (AUC) to measure performance of a given method.

Training on long sequences The typical duration of shots and the sampling rate imply a length of $\sim 1 \times 10^4$ samples per shot. We approximate the computation of the gradient of the loss with respect to the model parameters by truncated backpropagation through time [49]. We feed "chunks" of $T_{RNN} = 128$ timesteps at a time to the RNN. The gradients are then computed over this subsection, the internal states are saved, and then the next chunk is fed to the RNN while using the last internal states from the previous chunk as the initial internal states of the current chunk. This allows the RNN to learn long term dependencies while truncating the gradient backpropagation through time to T_{RNN} time steps.

Mini-batching Mini-batching [50] is an important technique for improving GPU performance [51] and accelerating training convergence of DL models. The gradients of the loss with respect to the parameters are computed for several examples in parallel and then averaged. For this to work efficiently, the architecture for the forward and backward pass of each gradient computation needs to be equal for all the examples computed in parallel. This is not possible if different training examples have different lengths. Thus, training on sequences with diverse lengths is a significant and open problem for many sequence based learning tasks [49], particularly for sequences of vastly differing lengths. The traditional approach of bucketing [52, 49] would not work in our case because the sequence length is strongly correlated with whether shots are disruptive or non-disruptive and thus individual batches would be biased.

We implement a custom solution based on resetting the internal state of *individual examples* within a mini batch, as illustrated in figure S1 in the supplementary information. Since there is a persistent internal state between successive chunks in time, it is not possible to use more than one chunk from a given shot in a given mini-batch (chunks that are successive in the shot must also be presented to the RNN in successive mini-batches during training such that the internal state can be persisted correctly).

To train batch wise with a batch size of M, we need M independent (i.e. stemming from different shots) time slices of equal length to feed to the GPU. We do this by maintaining a buffer of M separate shots. At every training step, the first T_{RNN} time slices of the buffer are fed as the next batch. The buffer is then shifted by T_{RNN} time steps. Before adding shots to the buffer, they are cut at the beginning to be a multiple of T_{RNN} steps. Every time a shot is finished in the buffer (e.g. the light green shot in the figure S1), a new shot is loaded (dark green) and the RNN internal states of the corresponding batch index are reset for training. It is this ability of resetting the internal state of select batch indices that allows batch wise training on shots of varying lengths. The internal states of the other batch indices are maintained and only reset when a new shot is begun in their respective index of the buffer. Thus, the internal state is persisted during learning for the entire length of any given shot. This allows the RNN to learn temporal patterns much longer than the unrolling length T_{RNN} and potentially as long as the entire shot. The random offsets of the shots against each other and random shuffling of the trainig set provides a mixture of disruptive and non-disruptive samples for the network at every batch to stabilize training. The fetching of shots and filling of the buffer is performed in a separate computational thread to pipeline neural network training work with data loading work.

Hyperparameters Overall, the data normalization, training procedure, and model architecture produce a large number of hyperparameters that must be tuned in order to maximize predictive performance. These hyperparameters include numerical values such as the learning rate and the number of LSTM layers, but also more abstract categorical variables such as the precise model architecture or the normalization algorithm used for different signals. We summarize these parameters in extended data table 3.

Throughout this work and for each data set, the “best” model is found by hyperparameter tuning. This is done by random search in the respective hyperparameter space of each method, i.e. by training a number of models with random hyperparameters on the training set and choosing the one with highest performance on the validation set [53]. In all our tests, gradient boosted trees [27] performed best among classical models, leading to the results in figure 2, extended data figure 1, as well as table 1. All DL models are trained with early stopping using the validation AUC as the metric, with a patience of 3 epochs [54]. The best performing models for table 1 of the main text are obtained in this way by using 20 random trials for each method.

Experimenting with a small number of shots from the test machine To simulate the scenario of being able to run a few disruptive shots on the test machine for cross-machine prediction, we remove a set δ of shots from the testing set on the "big" machine (JET) by sampling random shots until a fixed number of 5 disruptive shots have been sampled. In our experiment, δ contains 5 disruptive shots and 16 non-disruptive shots. The training and validation data from the "small" machine (DIII-D) are augmented with this set δ to have both more accurate training and a better measure of validation performance, and the best cross-machine model is re-trained without extra tuning. Moreover, we apply to particular importance-weighting or loss adjustment for these extra shots. It is possible that the positive effect of the additional shots could be even further enhanced by such methods. The numbers reported in table 1 are generated using this procedure.

We also tested the same scenario by sampling shots chronologically instead of randomly from the testing set for the same hyperparameters. The idea behind this approach is that this may more closely resemble the true distribution of shots that one would have access to during a new campaign on a new machine. We found that this approach did not change the results significantly beyond the generally expected stochastic fluctuations in AUC values of order ± 0.01 (which are due to random training and parameter initializations). The overall ordering of methods and qualitative range of performance remained the same.

Finally, we also performed tests with numbers of disruptive shots different than 5. While some stochastic fluctuations are as always expected, we find that performance generally increases monotonically for 0 to 7 shots, and saturates after about 7 disruptive shots. Increasing the number of disruptive shots also improves the fraction of models (given randomly chosen hyperparameters) that converge to strong cross-machine performance during training. Since the shots used are removed from the testing set on which the method is ultimately evaluated, it is not possible with this approach to make a fair comparison of performance for large numbers of removed shots, as the testing set would become significantly different.

Training for classical models Training on large data sets is problematic for classical methods, since training algorithms often do not scale well to high performance computing (HPC) environments. SVMs for example have a training cost quadratic in the number of examples [55] which makes very large data sets infeasible. Additionally, parallel algorithms for training single models across many worker nodes are lacking. We use a similar approach to that used in the literature [10] of producing features for training of the classical ML models in this study. At every time step, features are extracted for each signal from a time window comprising the last 32 ms. Since classical methods cannot learn to automatically extract patterns of various temporal scales from arbitrary sequence lengths, this window size represents a manually tuned tradeoff between detecting long and short temporal patterns that might be relevant for disruption prediction. For each time window of each signal we compute the mean, maximum, and standard deviations, as well as the 4 parameters of a 3rd order polynomial fit. Thus, for n signals, we have a $7n$ dimensional feature vector at every time step. We then train the models by considering each time step a separate "training example". We train on a random subset of 10^6 such examples to avoid prohibitively long training times. The target value is the same as in the "hinge" target for the DL model (i.e. -1 or 1). We implemented random forests, SVMs with linear and nonlinear kernels, multi-layer perceptrons with a single hidden layer, and gradient boosted trees. All classical ML models are implemented in Scikit-Learn [56] and we use XGBoost [27] to provide the functionality for the gradient boosted trees.

Distributed training In our code, we use python multiprocessing to parallelize preprocessing, shot loading, downloading and basically all components of the preparation and training pipeline. The vast majority of the computational load however occurs during the model training phase. While effective massive-scale parallelization of neural network training is an important open research question [57, 32], the idea of data-parallel training is already being used for the largest and most advanced DL models to date [58]. Most state of the art industrial algorithms [47, 59] use a parameter server approach with centralized communication paradigms. By contrast, our MPI implementation allows us to take advantage of highly optimized divide-and-conquer communication routines with logarithmic scaling in the number of processes. As communication is often the bottleneck in distributed training systems, efficient implementation of this component of the training algorithm is key. We empirically observe a very high ratio of computation to communication time ($> 95\%/5\%$) during distributed training, even on many dozens of GPUs.

The distributed training sequence can be described as follows (as illustrated in figure S2 in the supplementary information):

1. N models are run with their own copy of the current parameters (W);
2. Each computes a gradient step on a different subset (mini-batch) of the data;
3. The gradients are reduced (averaged), and the updates are made to a global set of parameters;
4. The new global parameters are broadcast to the N models; and

5. Efficient communication is achieved using a custom MPI implementation.

This effectively amounts to training with a large batch size that is the original batch size $N_{examples/batch}$ multiplied with the number of workers $N_{examples/batch} \to N_{worker} \times N_{examples/batch}$. To actually achieve a speedup for training, we then multiply the learning rate by N_{worker}. This means the algorithm is taking fewer learning steps, but each step is larger in magnitude and has smaller variance (since it is based on more data due to the larger batch size).

Our parallelized MPI implementation is also used for massively parallel batch wise inference which speeds up the computation of validation metrics between training epochs. To run batch wise inference, all shots are padded in the end with zeros to be of the same length. Since information enters the RNN only causally, these paddings do not influence the computation in the earlier sections of the shot can then simply be cut off to obtain the final shot output.

Scaling studies

The experiments illustrated in figure 3 (b) were performed on the Titan supercomputer [60], and we have replicated these scaling results on both the TSUBAME 3.0 and OLCF Summit supercomputers [61, 62]. The hyperparameter tuning experiment described in figure 3 (c) in the main text, engaging 10^4 GPUs by training 100 models in parallel, each using 100 GPUs, was also conducted on the Titan supercomputer and on the JET data set. The 1 and 100 GPU scenarios are fictitious since the time required to actually run these scenarios would have been prohibitively large. The ratio of time required to train a single model using 1 instead of 100 GPUs was estimated using scaling data as in figure 3 (a) from the main text. Specifically, the estimate is obtained by comparing timings between 4 and 100 GPUs and extrapolating from there down to 1, since 4 is the smallest machine architecture that is equal in configuration to 100 GPUs (since each node has 4 GPUs). The scenario of training the 100 models in serial (one at a time) was modeled by considering a large number ($5 \cdot 10^3$) of randomized serial arrangements of the 100 already recorded recorded runs, extracting results (such as the time required to find a model of a certain validation AUC) from each of those fictitious re-orderings, and averaging the results over all arrangements.

Figure 3 (c) in the main text shows some initial indications that convergence patterns are changing when using 256 GPUs or more. While it is known that deep neural networks become harder to train to full accuracy with many worker GPUs [32] — which corresponds to very large batch sizes — we expect that with larger models (in terms of trainable parameters), larger data sets, and higher-dimensional signals, even greater parallelism than reported in figure 3 (c) and the associated sections in the main text will become practical for single model training. Moreover, promising recent techniques such as learning rate warm-up, scaling, or cycling [32, 63] will likely further extend the practical range of parallelism as well — thus further engaging our code's capability of scaling to 1000s of GPUs.

Signal importance studies

In order to prioritize investments in higher quality data acquisition, and also in order to gain new scientific/physics insights, it is important to quantify the importance of the various signals for the predictability of disruptions. To this end, we train a model with just a single signal at a time and measure the final prediction performance, as seen in extended data figure 2 (a). This is then a proxy for the disruption relevant information contained in the respective signal. We also train a model with all signals but a single signal left out and give results in extended data figure 2 (b). By comparing the performance to a model trained on all signals (green), the relative drop in performance is a measure of how important that signal was for the full model. Naturally, a model trained on many signals might incorporate high-order interactions between signals, whose effects are not well measured by either of these two approaches. Moreover, the results are stochastic and vary according to model instantiations (due to random training initialization) and hyperparameters. Thus, these estimates should only be seen as a first-order measure of signal importance. Since these studies require training and testing several models in parallel as for hyperparameter tuning, they again can be sped up significantly using HPC.

Code Availability

The code used in this work is open source and available at [45].

References

[1] Mote Jr, C., Dowling, D. A. & Zhou, J. The power of an idea: The international impacts of the grand challenges for engineering. *Engineering* **2**, 4–7 (2016).

[2] Schuller, F. Disruptions in tokamaks. *Plasma Physics and Controlled Fusion* **37**, A135 (1995).

[3] De Vries, P. *et al.* Requirements for triggering the iter disruption mitigation system. *Fusion Science and Technology* **69**, 471–484 (2016).

[4] Lehnen, M. *et al.* Disruptions in iter and strategies for their control and mitigation. *Journal of Nuclear materials* **463**, 39–48 (2015).

[5] Tang, W. *et al.* Scientific grand challenges: Fusion energy science and the role of computing at the extreme scale. *PNNL-19404* 212 (2009).

[6] Aymar, R., Barabaschi, P. & Shimomura, Y. The iter design. *Plasma physics and controlled fusion* **44**, 519 (2002).

[7] Wroblewski, D., Jahns, G. & Leuer, J. Tokamak disruption alarm based on a neural network model of the high-beta limit. *Nuclear Fusion* **37**, 725 (1997).

[8] Cannas, B., Fanni, A., Marongiu, E. & Sonato, P. Disruption forecasting at jet using neural networks. *Nuclear fusion* **44**, 68 (2003).

[9] Murari, A. *et al.* Prototype of an adaptive disruption predictor for jet based on fuzzy logic and regression trees. *Nuclear Fusion* **48**, 035010 (2008).

[10] Vega, J. *et al.* Results of the jet real-time disruption predictor in the iter-like wall campaigns. *Fusion Engineering and Design* **88**, 1228–1231 (2013).

[11] Windsor, C. *et al.* A cross-tokamak neural network disruption predictor for the jet and asdex upgrade tokamaks. *Nuclear fusion* **45**, 337 (2005).

[12] Luxon, J. L. A design retrospective of the diii-d tokamak. *Nuclear Fusion* **42**, 614 (2002).

[13] Matthews, G. *et al.* Jet iter-like walloverview and experimental programme. *Physica Scripta* **2011**, 014001 (2011).

[14] Freidberg, J. P. *Plasma physics and fusion energy* (Cambridge university press, 2008).

[15] See the supplementary information.

[16] Taylor, P. *et al.* Disruption mitigation studies in diii-d. *Physics of Plasmas* **6**, 1872–1879 (1999).

[17] Tang, W. M. & Chan, V. Advances and challenges in computational plasma science. *Plasma physics and controlled fusion* **47**, R1 (2005).

[18] De Vries, P. *et al.* Survey of disruption causes at jet. *Nuclear fusion* **51**, 053018 (2011).

[19] Carleo, G. & Troyer, M. Solving the quantum many-body problem with artificial neural networks. *Science* **355**, 602–606 (2017).

[20] Carrasquilla, J. & Melko, R. G. Machine learning phases of matter. *Nature Physics* **13**, 431 (2017).

[21] Throughout this paper we will refer to ML models that do not apply DL paradigms as "classical" algorithms. They have also been termed "shallow learning" in the literature to explicitly distinguish them from deep learning approaches [28].

[22] Rattá, G., Vega, J., Murari, A., Johnson, M. & Contributors, J.-E. Feature extraction for improved disruption prediction analysis at jet a. *Review of scientific instruments* **79**, 10F701 (2008).

[23] Rattá, G., Vega, J., Murari, A., Contributors, J.-E. *et al.* Improved feature selection based on genetic algorithms for real time disruption prediction on jet. *fusion Engineering and Design* **87**, 1670–1678 (2012).

[24] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

[25] Bradley, A. P. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition* **30**, 1145–1159 (1997).

[26] Liaw, A., Wiener, M. *et al.* Classification and regression by randomforest. *R news* **2**, 18–22 (2002).

[27] Chen, T. & Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794 (ACM, 2016).

[28] Chollet, F. *Deep learning with Python* (Manning Publications, 2018).

[29] Barton, J. E., Wehner, W. P., Schuster, E., Felici, F. & Sauter, O. Simultaneous closed-loop control of the current profile and the electron temperature profile in the tcv tokamak. In *American Control Conference (ACC), 2015*, 3316–3321 (IEEE, 2015).

[30] Tobias, B. *et al.* Commissioning of electron cyclotron emission imaging instrument on the diii-d tokamak and first data. *Review of Scientific Instruments* **81**, 10D928 (2010).

[31] De Vries, P., Johnson, M., Segui, I. & Contributors, J. E. Statistical analysis of disruptions in jet. *Nuclear Fusion* **49**, 055011 (2009).

[32] Goyal, P. *et al.* Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[33] Svyatkovskiy, A., Kates-Harbeck, J. & Tang, W. Training distributed deep recurrent neural networks with mixed precision on gpu clusters. In *Proceedings of the Machine Learning on HPC Environments*, 10 (ACM, 2017).

[34] Top500 supercomputers. Available at `https://www.top500.org/lists/2017/11/` (2018/01/11).

[35] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529 (2015).

[36] Coelho, R. *et al.* Synthetic diagnostics in the european union integrated tokamak modelling simulation platform. *Fusion Science and Technology* **63**, 1–8 (2013).

[37] Comparable with some of the largest published ML data sets [42].

[38] Zavaryaev, V. *et al.* Fusion physics. In Kikuchi, M., Lackner, K. & Tran, M. Q. (eds.) *Plasma Diagnostics*, chap. 4, 360–534 (International Atomic Energy Agency, 2012).

[39] Ferron, J. *et al.* Real time equilibrium reconstruction for tokamak discharge control. *Nuclear Fusion* **38**, 1055 (1998).

[40] Alonso, J. *et al.* Fast visible camera installation and operation in jet. In *AIP Conference Proceedings*, vol. 988, 185–188 (AIP, 2008).

[41] Zadrozny, B., Langford, J. & Abe, N. Cost-sensitive learning by cost-proportionate example weighting. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, 435–442 (IEEE, 2003).

[42] Deng, J. *et al.* Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255 (IEEE, 2009).

[43] Maas, A. L. *et al.* Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, 142–150 (Association for Computational Linguistics, 2011).

[44] Marcus, M. P., Marcinkiewicz, M. A. & Santorini, B. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* **19**, 313–330 (1993).

[45] Kates-Harbeck, J. & Svyatkovskiy, A. FRNN Codebase. `github.com/PPPLDeepLearning/plasma-python` (2017).

[46] Chollet, F. *et al.* Keras. `https://github.com/fchollet/keras` (2015).

[47] Abadi, M. *et al.* Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[48] Multi-dimensional signals are counted as one signal, since they should *not* be normalized individually. Gradient information is important in such signals and would be distorted if each channel was normalized individually.

[49] Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[50] Dean, J. *et al.* Large scale distributed deep networks. In *Advances in neural information processing systems*, 1223–1231 (2012).

[51] Chetlur, S. *et al.* cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[52] Khomenko, V., Shyshkov, O., Radyvonenko, O. & Bokhan, K. Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. In *Data Stream Mining & Processing (DSMP), IEEE First International Conference on*, 100–103 (IEEE, 2016).

[53] Note that the validation set is from the same distribution as the training set, since we assume that a real application would not have access to any data from the testing set at training time. Thus, hyperparameter tuning might not find the truly best model, since the optimization metric is performance on the validation set and not the test set itself.

[54] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[55] Chang, C.-C. & Lin, C.-J. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* **2**, 27 (2011).

[56] Pedregosa, F. *et al.* Scikit-learn: Machine learning in python. *Journal of machine learning research* **12**, 2825–2830 (2011).

[57] Das, D. *et al.* Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709* (2016).

[58] Wu, R., Yan, S., Shan, Y., Dang, Q. & Sun, G. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876* **7** (2015).

[59] Chen, T. *et al.* Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[60] The Titan Supercomputer. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/`. Accessed: 2018-04-02.

[61] Morgan, T. P. Japan Keeps Accelerating With Tsubame 3.0 AI Supercomputer. `https://www.nextplatform.com/2017/02/17/japan-keeps-accelerating-tsubame-3-0-ai-supercomputer/`. Accessed: 2018-04-02.

[62] The Summit Supercomputer. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/`. Accessed: 2018-04-02.

[63] Smith, L. N. Cyclical learning rates for training neural networks. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, 464–472 (IEEE, 2017).

[64] We define the "best" threshold as the value that maximizes the quantity $TP - FP$ where TP is the true positive rate. This is equivalent to finding the point on the ROC curve furthest in the "northwest" direction.

[65] We define the "best" threshold as the value that maximizes the quantity $TP - FP$ where TP is the true positive rate. This is equivalent to finding the point on the ROC curve furthest in the "northwest" direction.

[66] The only indication otherwise is that a measurable but only slim and possibly stochastic increase in performance is observed when training a model without $I_{p,direct}$ or $I_{p,error}$.

[67] The only indication otherwise is that a measurable but only slim and possibly stochastic increase in performance is observed when training a model without $I_{p,direct}$ or $I_{p,error}$.

[68] Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[69] Gentile, C. & Warmuth, M. K. Linear hinge loss and average margin. In *Advances in Neural Information Processing Systems*, 225–231 (1999).

Acknowledgments

Funding: JKH was supported by the Department of Energy Computational Science Graduate Fellowship Program of the Office of Science and National Nuclear Security Administration in the Department of Energy under contract DE-FG02-97ER25308. AS is supported by Princeton University's Institute for Computational Science & Engineering (PICSciE). WT is supported by the DOE Princeton Plasma Physics Laboratory (PPPL) as well as PICSciE.

DIII-D Funding Statement: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Award DE-FC02-04ER54698. DIII-D data shown in figures in this paper can be obtained in digital format by following the links at `https://fusion.gat.com/global/D3D_DMP`. **Disclaimer:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

EUROfusion Funding Statement: This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Author Contributions: JKH conceived the idea and wrote the code to apply deep learning to disruption prediction, including the HPC and MPI features, curated the data sets, ran and analyzed computational experiments, and generated the theoretical scaling analysis. With input from his co-authors and collaborators, he produced the figures. He also wrote and edited the manuscript and supplementary information. AS extended and co-authored the code base, ran computational experiments including the initial deployment of the code on supercomputers, and edited the paper. WT is the primary investigator who supervised and supported the implementation of the project at all stages and edited the paper. He also initiated the collaborations with JET, and established partnerships with leading supercomputing facilities to carry out the DL studies at scale.

Acknowledgements: We are grateful to Eliot Feibush of PPPL and PICSciE for supporting our visualization and data collection efforts. We also thank Kyle Felker of Princeton U.; Tony Donne, Emmanuel Joffrin, Mikhail Maslov, Andrea Murari, and Jesus Vega of JET; Richard Buttery and Ted Strait of General Atomics; Robert Granetz and Cristina Rea of MIT; Matthew

Parsons, Nik Logan, Raffi Nazikian, and Michael Churchill of PPPL; and Peter DeVries of ITER for their support and for valuable discussions. We also express our profound gratitude to the JET Contributors (See the author list of “X. Litaudon et al 2017 Nucl. Fusion 57 102001”) and their management as well as to General Atomics (GA) and its DIII-D tokamak project for the generous access to their fusion databases.

Data and Materials Availability: The data for this study has restricted access with permission required from the EUROfusion and GA management. All code is open-source [45].

Competing Interests: The authors declare no competing interests.

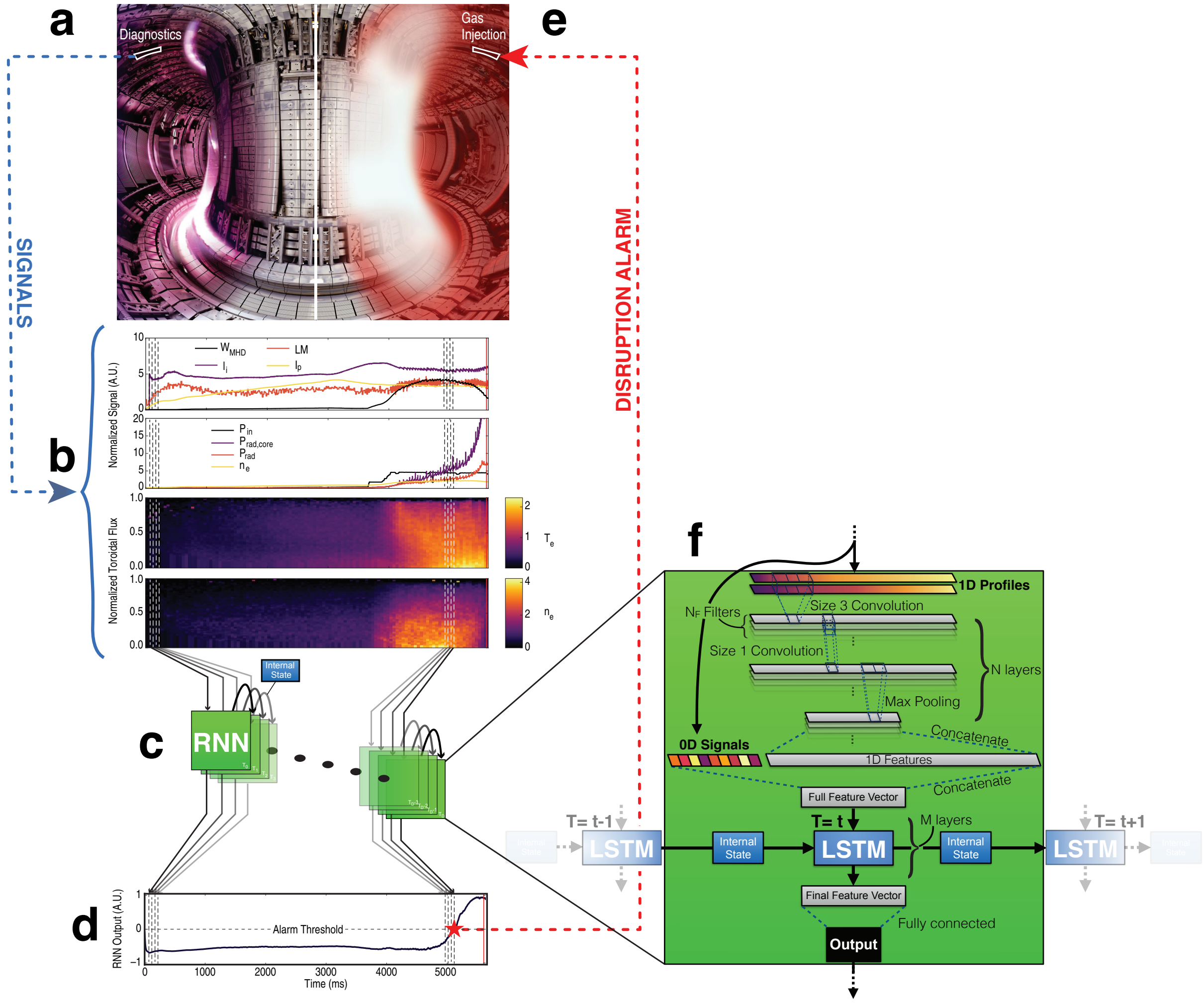


Figure 1: **System overview and disruption prediction workflow.** The top image shows an interior view of the JET tokamak with a non-disruptive plasma on the left and a disruptive plasma on the right. Diagnostics **(a)** provide streams of sensory data **(b)** which are fed to the recurrent neural network (RNN) based deep learning algorithm **(c)** every 1 ms, producing a corresponding "disruptivity" output at every time step **(d)**. If the output crosses a preset threshold value (dashed horizontal line), a disruption alarm is called (red star). This alarm triggers mitigation action such as gas injection **(e)** into the tokamak to reduce the deleterious effects of the impending disruption. A detailed schematic of our deep learning model is shown in **(f)**. The input data consists of scalar 0D signals and 1D profiles. N layers of convolutional (N_F filters each) and downsampling (max-pooling) operations reduce the dimensionality of the profile data and extract salient low dimensional representations (features). These features are concatenated with the 0D signals and fed into a multilayer long-short term memory network (LSTM) with M layers, which also receives its internal state from the last time step as input. The resulting final feature vector ideally contains salient information from the past temporal evolution and the present state of all signals. This vector is fed through a fully connected layer to produce the output.

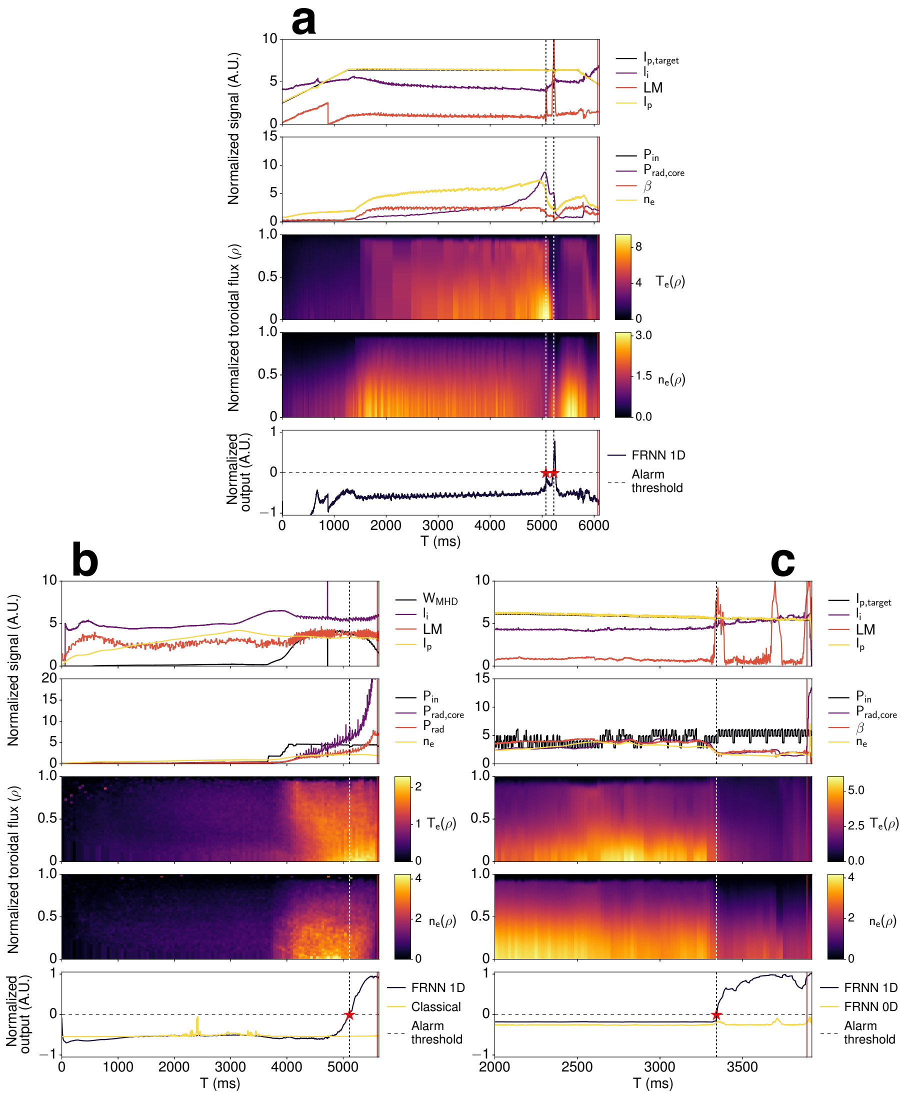


Figure 2: **Example predictions on real shots from DIII-D (a,c) and JET (b).** For each shot, the top two panels show scalar signals, the next two panels show the electron temperature and density profiles, respectively, and the bottom panel shows the model output as a function of time. $T = 0$ is defined as the first time point for which all signals are present in the data base, which can differ from the standard DIII-D and JET time base. Only a representative subset of the signals used by the algorithm is plotted and each signal is shown in its normalized form (see methods for details and see extended data table 1 for descriptions of each signal). The red stars and the dashed vertical lines indicate alarms. Disruptive shots **(b-c)** have a vertical red line at the 30 ms deadline before the disruption. **(a)** DIII-D shot 148778: a false alarm is triggered about 5200 ms into the shot by a minor disruption. Careful inspection reveals in fact two separate minor disruptions in close succession, corresponding to the spikes in the output and the resulting alarms. **(b)** JET shot 83413: the slow rise in radiated power allows our deep learning (DL) approach (black) to correctly predict the disruption hundreds of ms in advance, which is missed by the best classical model (yellow, see main text for details). **(c)** DIII-D shot 159593: only the DL model with access to profile information (black) can correctly predict this oncoming disruption, while it is missed by the model trained on just scalar signals (yellow).

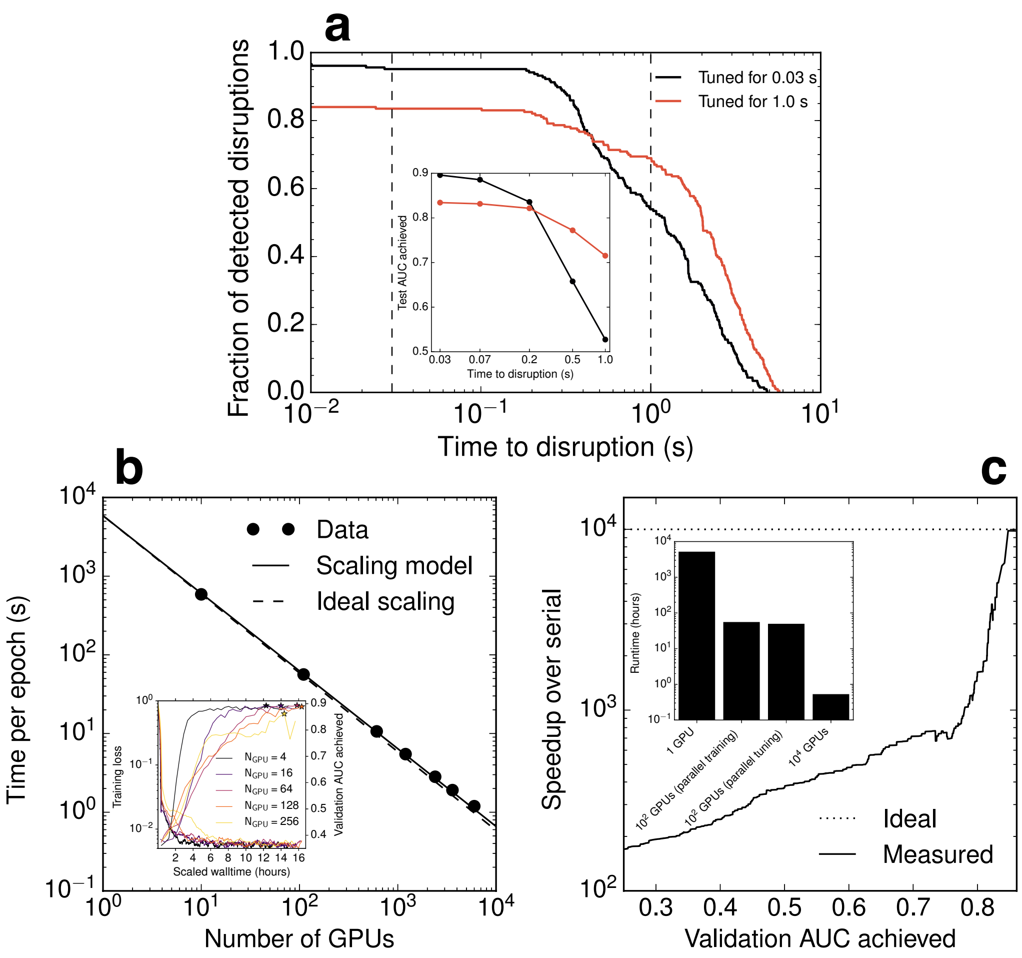
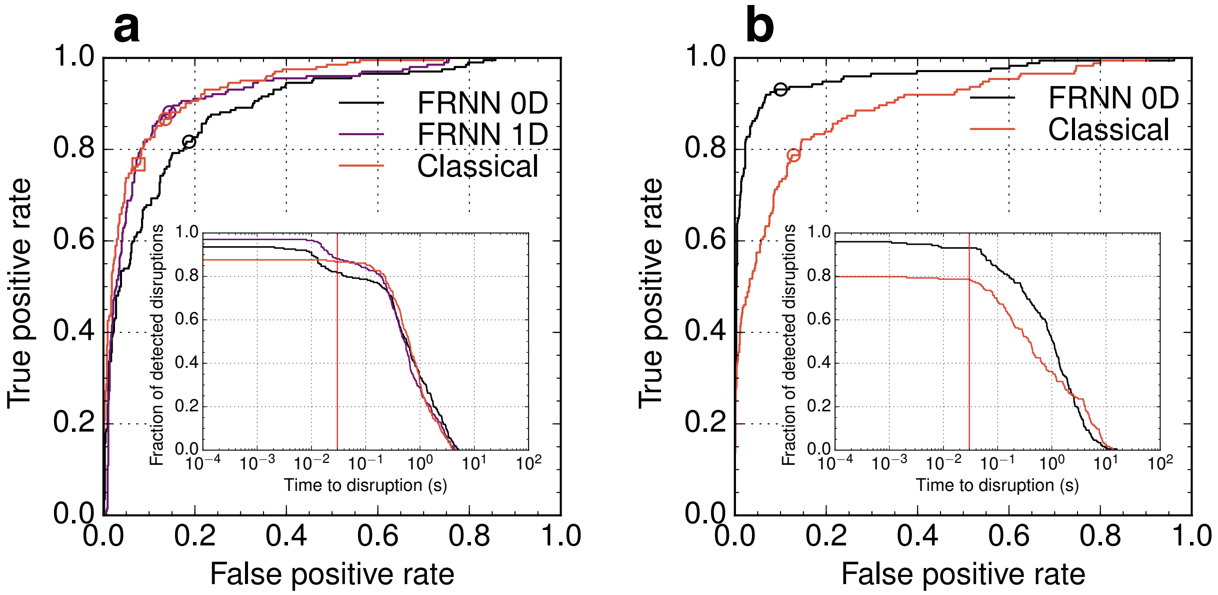


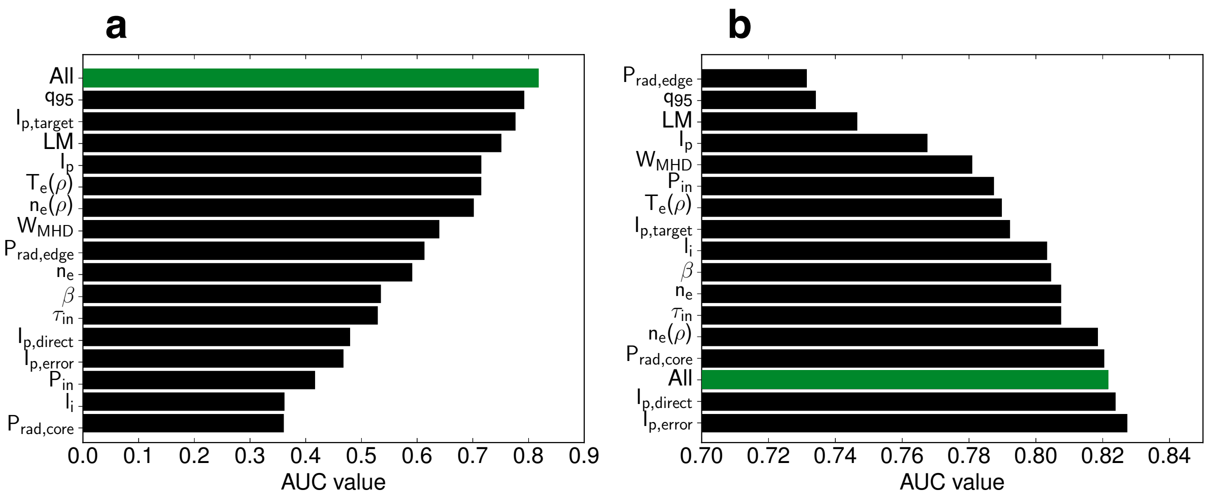
Figure 3: **High performance computing results.** **(a)** Accumulated fraction of detected disruptions and area under the ROC curve (AUC) values achieved (inset) on DIII-D as a function of time to disruption for two models optimized for performance at a time to disruption of 30 (black) and 1000 ms (orange), respectively. **(b)** Time required for completing one pass over the data set (one "epoch") during training vs number of GPUs engaged. Experimental results (circles) are compared with a semi-empirical theoretical scaling model (solid line [15]) and ideal scaling (dashed line). The inset shows actual training progress measured via mean training loss (i.e. the difference between target and realized output of the model; decreasing curves) and validation AUC (increasing curves) for various numbers of GPUs as a function of scaled walltime (walltime $\times N_{GPUs}$). The best validation AUC value is denoted by a star. The 256 GPU run shows some initial indications that the pattern of convergence is changing, though still giving final testing AUC as good as the other runs. **(c)** Results for hyperparameter tuning with 10^4 GPUs with parallel random search across 100 models, trained on 100 GPUs each. The time to solution for finding a model of given validation AUC is compared to the scenario of using a single GPU. The inset shows the time required for finding the best model in the scenarios of using 1, 10^2, and 10^4 total GPUs. For 100 GPUs, we distinguish training the 100 models in serial, but using 100 GPUs for each model ("parallel training") vs. running 100 models in parallel, trained on 1 GPU each ("parallel tuning"), which both achieve nearly 100x speedup.

	Single machine		Cross-machine		Cross-Machine with "glimpse"
Training Set	DIII-D	JET (CW)	JET (CW)	DIII-D	DIII-D $+\delta$
Testing Set	DIII-D	JET (ILW)	DIII-D	JET (ILW)	JET (ILW) $-\delta$
Best classical Model	**0.937**	0.893	0.636	0.616	0.851
FRNN 0D	0.890	**0.952**	**0.761**	0.817	0.879
FRNN 1D	0.922	–	–	**0.836**	**0.911**

Table 1: **Prediction results.** Test set performance of the best models measured as area under the ROC curve (AUC) at 30 ms before the disruption. We compare FRNN with and without profile information ("1D" and "0D", respectively), and the best classical approach. The best model for each data set is shown in bold. The last column shows results for cross-machine testing with a small amount of data δ from the testing machine added to the training set (see text for details). 1.0 is perfect performance and 0.5 is equivalent to random guessing. Since the relevant diagnostic for 1D profiles was not available on most JET shots from the carbon wall data set, 1D profiles are not included when training on JET data.



Extended Data Figure 1: **ROC curves on the test set for our model and the best classical model for DIII-D (a) and JET (b).** The areas under the curves correspond to the values in table 1. The inset shows the fraction of detected disruptions as a function of the time to disruption for an "optimal" threshold value. On the corresponding ROC curve of the same color, this optimal threshold defines a point which is indicated with a circle (see main text for details). The inset also shows the 30 ms detection deadline as vertical red line. While in (a) the area under the ROC curve (AUC) is slightly higher for the classical method (see table 1), FRNN performs equally well in the interesting "upper left" region of high true positives (TPs) and low false positives (FPs). Moreover, only our approach provides additional detections between 30 and 10 ms to the disruption, reacting to the spikes in radiated power which often occur on this time scale before the disruption (see $P_{rad,\text{core}}$ in figure 2 (c)). Thus FRNN could provide improved predictive performance if mitigation technology becomes faster in the future. In addition, a threshold value in practice needs to be selected for calling alarms. The best threshold value is estimated by optimizing it [65] on the *training set*, in hopes that it will still perform well on the unseen testing set. For FRNN, the threshold generalizes excellently (black and purple circles). For the classical approach, while the overall ROC curve is encouraging, the threshold estimate is poor (orange square) and far from its ideal position (orange circle). For each method, the fraction of detected disruptions are shown in the inset as a function of time until disruption by using the threshold value corresponding to the circle positions — which for the classical method we determine manually with knowledge of the testing set (to give a conservative and maximally favorable estimate of its performance). Median alarm times are $\sim 500 - 700$ ms on DIII-D and $\sim$ 1000 ms on JET. Encouragingly, a majority of disruptions are detected with large warning times of hundreds of ms — sufficient for disruption mitigation (requiring $\sim$ 30 ms) and key for possible future preventative plasma control without the need for shut down.



Extended Data Figure 2: **Signal importance studies.** Signals are ordered from top to bottom in decreasing order of importance. Models were trained on the DIII-D data set. (a) Test set area under the ROC curve (AUC) values achieved by models trained on a single signal at a time. The AUC value is representative of how much information is contained in that single signal. We also show the performance for a model trained on all signals for comparison (green bar). (b) Test AUC values for a model trained on all signals except the labeled one. In this case, the *drop* in performance compared to the model trained on all signals (green bar) is a measure of how important the given signal is for the final model. The exact results for both figures are in general stochastic and vary over hyperparameters and for each new training session, so only general trends should be inferred. It appears consistently that the locked mode, plasma current, radiated power, and q_{95} signals contain a large amount of disruption-relevant information, similar to the results of past studies of signal importance on JET [23]. Both panels — in particular figure (a) which measures the information content of a single signal at a time — also confirm that there is significant information in the profile signals. With higher quality reconstructions, more frequent sampling, and better (causal) temporal filtering (to obviate the need to shift the signal in time and thus lose time sensitive information) they are likely to become even more relevant. This gives faith that there is much useful information in higher dimensional data that should be considered in the future. Another benefit of the DL approach is that we can use most signals without having to worry too much about confusing the algorithm and reducing performance [67], and thus without having to spend much time on signal selection. For other algorithms, signal selection (e.g. removing correlated, noisy, or non-informative signals) is key [23].

Signal description	Avail. on machines	Symbol	Numerical scale DIII-D	Numerical scale JET
Plasma current	DIII-D, JET	I_p	3.81×10^{-1} MA	5.03×10^{-1} MA
Plasma current target	DIII-D	$I_{p,}$target	3.93×10^{-1} MA	–
Plasma current error	DIII-D	$I_{p,}$error	3.10×10^{-2} MA	–
Plasma current direction	DIII-D	$I_{p,}$direct	1.0	–
Internal inductance	DIII-D, JET	l_i	2.02×10^{-1}	1.51×10^{-1}
Electron temperature profile	DIII-D,JET*	$T_e(\rho)$	9.53×10^{-1} keV	1.53×10^{3} eV
Electron density profile	DIII-D,JET*	$n_e(\rho)$	1.47×10^{19} m^{-3}	2.98×10^{19} m^{-3}
Electron density	DIII-D, JET	n_e	1.19×10^{19} m^{-3}	4.69×10^{19} m^{-3}
Input power	DIII-D , JET	P_{in}	1.85×10^{6} W	4.47×10^{6} W
Input beam torque	DIII-D	P_{in}	1.47 Nm	–
Radiated power (core)	DIII-D, JET	$P_{rad,}$core	4.58×10^{-4} MW	4.05×10^{4} W
Radiated power (edge)	DIII-D, JET	$P_{rad,}$edge	4.94×10^{-4} MW	2.72×10^{4} W
Radiated power (total)	JET	P_{rad}	–	2.22×10^{6} W
Plasma energy	DIII-D, JET	W_{MHD}	2.79×10^{5} J	1.20×10^{6} J
Safety factor (magnetic field pitch)	DIII-D, JET	q_{95}	1.0	1.0
Normalized plasma pressure (ratio of thermal to magnetic pressure)	DIII-D	β	6.91×10^{-3}	–
Locked mode (nonrotating instability) amplitude	DIII-D, JET	LM	1.14 μT	5.72×10^{-5} T
Normalized toroidal magnetic flux	DIII-D, JET	ρ	1.0	1.0

Extended Data Table 1: **Signals considered and availability on the machines.** Electron density and electron temperature profiles (denoted by an asterisk) are only available on the more recent JET campaigns (ITER-like wall). Each signal is normalized by its numerical scale before feeding to the machine learning algorithms. The numerical scale is computed as the median of the standard deviations of the signal for all shots (see methods for details).

Machine	Shot range	Total shots	Num disruptive	Note
JET train	$66027 - 79853$	2894	215	JET carbon wall campaigns C23-C27b
JET validate	$66027 - 79853$	1425	87	JET carbon wall campaigns C23-C27b
JET test	$81852 - 83793$	1191	174	JET ITER-like wall campaigns C28-30
DIII-D train	$125500 - 168555$	1734	407	DIII-D shots since 2006
DIII-D validate	$125500 - 168555$	853	197	DIII-D shots since 2006
DIII-D test	$125500 - 168555$	862	206	DIII-D shots since 2006

Extended Data Table 2: **Data sets used in this paper.** Shots are obtained from the respective machines. All shots that contain data for all signals are used. No shots are discarded for bad or abnormal data, or for being known testing shots or intentional disruptions.

Hyperparameter	Explanation	Best value
η	Learning rate	1.7×10^-5
γ	Learning rate decay per epoch	0.985
λ	Weighting factor for positive examples	16
N	Number of recurrent neural network (RNN) layers	2
n_{cells}	Number of RNN cells per layer	200
T_{RNN}	Number of timesteps through which backpropagation is run	128
RNN type	Type of RNN	Long short term memory network (LSTM)
n_{batch}	Batch size	128
$T_{warning}$	Time at which the target function for a disruptive shot becomes positive in ms	10000
Target	Type of target function	TTD (Hinge)
Normalizer	Type of normalization scheme	Divide by σ
dt	timestep	0.001 s
$T_{min,warn}$	minimal acceptable warning time	0.03s
Hide warning	Whether to hide the last $T_{min,warn}$ timesteps during training (these wouldn't be available in a real shot)	`False`
N_F	Number of convolutional filters	128
M	Number of convolutional layers	128
$L_{filters}$	Size of 1D convolutional filters	3
L_{pool}	Size of 1D max pooling	2
Optimizer	Stochastic optimization scheme	Adam [68]
Dropout	Dropout probability	0.1
L2 Regularization	Weight regularization of all layers	10^-2
Clip norm	Maximum norm of gradients	10

Extended Data Table 3: **Hyperparameters to be optimized together with explanations and well-performing values.** The "hinge" target is -1 before $T_{warning}$, then $+1$ for disruptive shots. It requires a hinge loss [69]. The normalization scheme referenced in the table in the "Normalizer" row divides each signal by its global numerical scale across the data set (see methods for details). While all quoted parameters perform well on both tokamaks, the specific values shown above are found by optimizing for validation performance on JET.