IAEA nuclear fusion

PAPER • OPEN ACCESS

# Fast dynamic 1D simulation of divertor plasmas with neural PDE surrogates

To cite this article: Yoeri Poels *et al* 2023 *Nucl. Fusion* **63** 126012

View the article online for updates and enhancements.

OPEN ACCESS
IOP Publishing | International Atomic Energy Agency
Nuclear Fusion

Nucl. Fusion **63** (2023) 126012 (28pp) https://doi.org/10.1088/1741-4326/acf70d

# Fast dynamic 1D simulation of divertor plasmas with neural PDE surrogates

**Yoeri Poels**[1,3,*], **Gijs Derks**[2,4], **Egbert Westerhof**[4], **Koen Minartz**[1], **Sven Wiesen**[5] **and Vlado Menkovski**[1]

[1] Eindhoven University of Technology, Mathematics and Computer Science, NL-5600MB Eindhoven, Netherlands
[2] Eindhoven University of Technology, Control Systems Technology, NL-5600MB Eindhoven, Netherlands
[3] École Polytechnique Fédérale de Lausanne, Swiss Plasma Center, CH-1015 Lausanne, Switzerland
[4] Dutch Institute for Fundamental Energy Research, NL-5612AJ Eindhoven, Netherlands
[5] Forschungszentrum Jülich GmbH, Institut für Energie- und Klimaforschung—Plasmaphysik, DE-52425 Jülich, Germany

E-mail: y.r.j.poels@tue.nl

Received 23 June 2023, revised 18 August 2023
Accepted for publication 5 September 2023
Published 25 September 2023

## Abstract

Managing divertor plasmas is crucial for operating reactor scale tokamak devices due to heat and particle flux constraints on the divertor target. Simulation is an important tool to understand and control these plasmas, however, for real-time applications or exhaustive parameter scans only simple approximations are currently fast enough. We address this lack of fast simulators using *neural partial differential equation (PDE) surrogates*, data-driven neural network-based surrogate models trained using solutions generated with a classical numerical method. The surrogate approximates a time-stepping operator that evolves the full spatial solution of a reference physics-based model over time. We use DIV1D, a 1D dynamic model of the divertor plasma, as reference model to generate data. DIV1D's domain covers a 1D heat flux tube from the X-point (upstream) to the target. We simulate a realistic TCV divertor plasma with dynamics induced by upstream density ramps and provide an exploratory outlook towards fast transients. State-of-the-art neural PDE surrogates are evaluated in a common framework and extended for properties of the DIV1D data. We evaluate (1) the speed-accuracy trade-off; (2) recreating non-linear behavior; (3) data efficiency; and (4) parameter inter- and extrapolation. Once trained, neural PDE surrogates can faithfully approximate DIV1D's divertor plasma dynamics at sub real-time computation speeds: In the proposed configuration, 2 ms of plasma dynamics can be computed in ≈0.63 ms of wall-clock time, several orders of magnitude faster than DIV1D.

Keywords: surrogate models, neural networks, machine learning, neural pde surrogates, scrape-off layer simulation, exhaust simulation, divertor plasma simulation

(Some figures may appear in colour only in the online journal)

[*] Author to whom any correspondence should be addressed.

1741-4326/23/126012+28$33.00 Printed in the UK

## 1. Introduction

Tokamak devices operate in a diverted plasma configuration to decrease the effect of plasma-wall interaction. In this configuration, open field lines rapidly transport particles leaking out of the core plasma to the divertor region. These particle and heat fluxes must be mitigated before they reach the divertor plates, as they can far exceed material limits if left uncontrolled [1–3]. However, mitigation techniques such as injecting impurities into the plasma edge have limits as they can degrade or can even be incompatible with core plasma performance [4, 5]. Consequently, fast and accurate simulation is crucial for understanding and controlling the behavior of the plasma in the divertor region. Recent modeling efforts, such as SOLPS-ITER [6], UEDGE [7], SD1D [8] and DIV1D [9], are showing great promise in simulating divertor plasmas on varying levels of fidelity. However, when aiming for real-time applications or for the examination of large parameter spaces, only simple analytical functions such as the two-point models [10] are fast enough, presenting a gap for fast high-fidelity simulation. In general, high-fidelity simulation is the cornerstone of model-based design and control for future reactors [11].

To enable real-time high-fidelity simulation of complex dynamics, machine learning-based surrogates are showing great potential. Recently, there has been an uptick in the development of artificial neural network (NN) based surrogate models for partial differential equation (PDE) solvers [12–16]. These *neural PDE surrogates* are trained in a data-driven manner: A classical numerical method first generates a dataset of PDE solutions, after which the NN is optimized to approximate the dynamics present in the dataset. At inference time, the NN can generate new solutions of a fidelity similar to the original solver, at a fraction of the computational cost.

Considering the above, we propose to use neural PDE surrogate models for fast and full-fidelity 1D simulation of divertor plasma dynamics. In this work we use DIV1D [9], a 1D dynamic model of the divertor plasma, as the physics-based model that generates the dataset of simulations.

The domain covers a 1D heat flux tube spanning from just below the X-point (upstream) to the divertor target; see figure 1 for an illustration. While the plasma behavior in DIV1D is simplified compared to higher fidelity codes such as SOLPS-ITER [6], a recent benchmark shows good agreement [9]. Moreover, DIV1D can simulate dynamically at a relatively low computational cost, allowing us to generate rich training datasets. The data represents a divertor plasma of the tokamak à configuration variable (TCV), with plasma conditions set around a recent evaluation of DIV1D w.r.t. SOLPS-ITER [9]. To evaluate the capabilities of a neural PDE surrogate, we vary the upstream parallel heat flux, the upstream plasma density, and the carbon concentration. Dynamics are induced by varying the upstream plasma density over time at various ramp rates. Additionally, we provide an initial investigation in (surrogate) modeling fast transients, simulated through fast spikes in the upstream heat flux and upstream plasma density.

Using this data we build surrogate models reproducing the full spatiotemporal solutions of DIV1D, specifically for a density ramp and a fast transient dataset. We build upon recent developments in neural PDE surrogates and tackle challenges such as long-rollout stability and efficiency. We evaluate a large number of methods and extend them to account for properties specific to these datasets, such as time-varying boundary conditions (BCs) and large variations in simulation length. We refer to the resulting surrogate model as DIV1D-NN, and evaluate it with respect to its simulation accuracy and its utility for downstream applications, e.g. detachment studies. For the latter, we evaluate properties and structures such as the approximate emission front [17], the target temperature given upstream conditions, and the reconstruction of phenomena such as a bifurcation in the target temperature [18] and roll-over in the target ion flux [19]. Moreover, we investigate the reproduction of fast transients somewhat resembling edge-localized modes (ELMs) [20]. All evaluations are done in relation to the computational cost of the proposed surrogate models.

In general, machine learning-based methods have been used in nuclear fusion research in various settings, for example in disruption prediction [21, 22], diagnostic processing [23, 24] or for accelerating simulation [25, 26]; see [27] for an exhaustive overview. Adjacent to our setting, NN-based surrogates have been proposed for accelerating scrape-off layer (SOL) simulation [28, 29]. For control purposes, [30] employ sparse regression techniques (SINDy [31]) on SOLPS-ITER simulations to identify reduced models of key boundary plasma quantities. In [32] an exploration is conducted for data-driven simulation of high-speed camera footage capturing MAST divertor dynamics (among other settings) using the Fourier Neural Operator [15]; in our case, we focus on surrogate modeling for simulators and consider a broader set of surrogates and evaluations. Closest to our setting is a data-driven surrogate for divertor plasmas using UEDGE [33]. They propose a fast NN-based surrogate for static prediction of divertor detachment. The surrogate maps upstream density, injected power, and carbon concentration to a set of synthesized diagnostics. In contrast, we use DIV1D to simulate the dynamical behavior of the divertor plasma, and we approximate the full spatiotemporal profile of the simulations. Modeling the divertor plasma as a dynamical system allows us to capture bifurcations in the solution [18], whereas static modeling (i.e. not taking into account previous equilibria) could be ill-suited for these phenomena.

The paper is organized as follows. Section 2 formalizes the problem setting. Section 3 describes the data generation procedure. Section 4 provides an overview of neural PDE surrogates and describes adaptations we make. The model is evaluated in section 5, and the applications and limitations are discussed. Finally, section 6 gives conclusions and an outlook. A summary of our contributions is as follows:

- We generate a dataset of dynamic simulations representing a realistic TCV divertor plasma, containing challenging nonlinear phenomena such as the roll-over in target ion flux and bifurcations in the target temperature. In addition, we generate a dataset for the exploratory analysis of reproducing transient events with surrogate models.

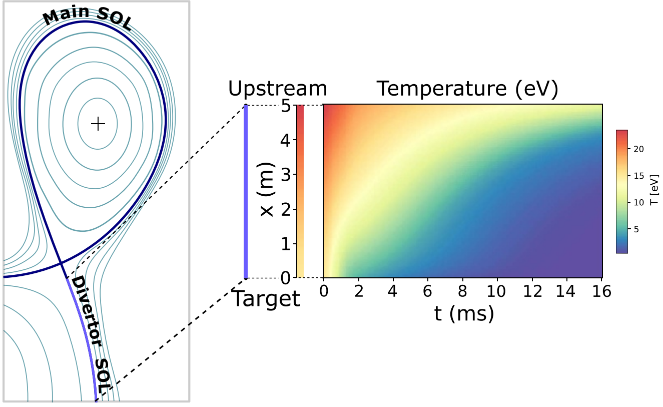


**Figure 1.** Simplified overview of the computational domain of DIV1D, visualized using an illustration of a TCV plasma. DIV1D models a 1D heat flux tube from just below the X-point up to the divertor target. The state of this tube is then evolved over time (illustrated here for the temperature only).

- We combine and implement a large number of state-of-the-art neural PDE surrogates in a common framework, with the purpose of building a dynamic surrogate model for divertor plasmas. We extend these methods to account for properties specific to the considered dynamics, such as the dependence on time-varying BCs and the wide range of simulation lengths, spanning two orders of magnitude between the shortest and longest simulated time.
- We conduct an in-depth evaluation of neural PDE surrogates and data properties, evaluating the following:
  * The trade-off between error and computation time.
  * Recovering higher-level properties and structures arising from non-linear behavior.
  * The efficiency with respect to the dataset size.
  * An analysis of inter- and extrapolation in the parameter space.
- We show that using neural PDE surrogates we can accurately approximate the dynamic behavior in the divertor plasma with up to 5 orders of magnitude speedup compared to DIV1D. As such, real-time use cases are within reach: DIV1D-NN generates 2 ms of plasma dynamics in ≈0.63 ms of wall-clock time. The surrogate captures high-level structures well, and can be trained using only hundreds of simulations. It shows strong performance within the evaluated parameter space. Global features of high-frequency transients can be reproduced with the proposed architecture.

## 2. Problem formulation

The target domain, the divertor plasma, is described through solutions of DIV1D, a 1D dynamic model of the scrape-off layer [9]. The DIV1D model solves 4 coupled time-dependent 1D fluid equations for the plasma density, plasma momentum, static plasma pressure and for neutral particles. These are solved along the divertor leg, where the computational domain extends from just below the X-point (upstream) to the target plate. For more details on the DIV1D model we refer to [9], with the settings used in this work described in section 3.

The PDE solutions are converted to plasma density ($n$), velocity ($v_{\|}$), temperature ($T$) and the neutral density ($n_{\mathrm{n}}$). These variables represent the quantities we simulate with a surrogate model. We denote the solution of all quantities with solution function $u(t,x)$, characterizing the solution as follows:

$$u(t,x)=\begin{bmatrix} n(t,x)\\ v_{\|}(t,x)\\ T(t,x)\\ n_{\mathrm{n}}(t,x)\end{bmatrix}, \qquad \begin{aligned} &t\in[0,t_{\max}],\\ &x\in\mathbb{X},\end{aligned} \tag{1}$$

where a simulation runs from time 0 to $t_{\max}$ over spatial domain $\mathbb{X}$. Consequently, the solution function $u\colon [0,t_{\max}]\times\mathbb{X}\to\mathbb{R}^4$ denotes a mapping from points in time and space to the four quantities.

In practice we operate on discretized solutions. To support neural PDE surrogates that assume fixed domains we use a fixed discretization for both the temporal and spatial domain. Spatial domain $\mathbb{X}$ is discretized to an equidistant grid with $Nx$ gridpoints over a length of $L$ (so $\mathrm{d}x=\frac{L}{Nx-1}$), denoted with $\mathbf{x}\in\mathbb{R}^{Nx}$. The temporal domain is discretized with fixed timestep $\mathrm{d}t$ to $Nt$ timesteps per simulation, denoted with $\mathbf{t}\in\mathbb{R}^{Nt}$. Consequently, a discretized solution is denoted as $\mathbf{u}^{\mathbf{t},\mathbf{x}}\in\mathbb{R}^{Nt\times Nx\times 4}$. Note that this discretization pertains to the data as used for the NN surrogate, not the settings used by DIV1D. The DIV1D solutions are downsampled and interpolated to the aforementioned discretization, for details on the numerics used with DIV1D we refer to section 3.

Intuitively, the data-driven surrogate modeling task boils down to mapping the varying conditions to the solution $\mathbf{u}^{\mathbf{t},\mathbf{x}}$. More precisely, we assume varying initial conditions and denote these as $\mathbf{u}^{0,\mathbf{x}}\in\mathbb{R}^{Nx\times 4}$. The varied BCs are denoted as $\mathbf{b}_{\mathrm{s}}\in\mathbb{R}^{Ns}$ and $\mathbf{b}_{\mathrm{d}}^{\mathbf{t}}\in\mathbb{R}^{Nt\times Nd}$ for $Ns$ static and $Nd$ dynamic BCs, respectively. Additional static conditions are denoted as $\mathbf{c}\in\mathbb{R}^{Nc}$, for $Nc$ such conditions. In the current work we do not consider additional spatial or time-varying constraints, although the methodology could easily be extended to such a setting. In summary, the goal is to learn the following mapping:

$$f_\theta(\mathbf{u}^{0,\mathbf{x}},\mathbf{b}_{\mathrm{s}},\mathbf{b}_{\mathrm{d}}^{\mathbf{t}},\mathbf{c})=\mathbf{u}^{\mathbf{t},\mathbf{x}}, \tag{2}$$

where $f_\theta$ denotes the to-be-learned function. Since the DIV1D equations are autonomous with respect to time, i.e. they do not directly depend on $t$, we can generate solutions by approximating a time-stepping operator with no explicit dependence on $t$. In other words, we parameterize an *autoregressive* model that evolves the state of the system with time $\mathrm{d}t$, denoted as follows:

$$\mathbf{u}^{t_{i-1},\mathbf{x}}+\mathrm{d}t\cdot f_\theta(\mathbf{u}^{t_{i-1},\mathbf{x}},\mathbf{b}_{\mathrm{s}},\mathbf{b}_{\mathrm{d}}^{t_i},\mathbf{c})=\mathbf{u}^{t_i,\mathbf{x}},\quad 0<i<Nt, \tag{3}$$

where $t_i$ denotes the $i$th element in $\mathbf{t}$. This formula is applied iteratively starting at $t_0=0$ to generate a full solution. Since $f_\theta$ is an NN, this formulation is much akin learning a neural ordinary differential equation [34] with an Euler solver with fixed timestep $\mathrm{d}t$. By predicting solutions in this form, the invariance to time is built into the model formulation, which should aid the surrogate model's performance. Additionally, generating solutions with an arbitrary number of timesteps is now possible, whereas in equation (2) function $f_\theta$ can only predict solutions up to a fixed horizon $Nt$.

## 3. Data generation

The data used in this work is generated using DIV1D. In [9], it is shown that the DIV1D model can characterize divertor plasma behavior over a range of upstream plasma densities with a single model setting. As such, the model forms a good starting point to generate data approximating the dynamics of the divertor plasma in TCV. However, we note that DIV1D does not self-consistently solve outside of the SOL, hence the generated data does not cover time-dependent interactions between the SOL and external domains.

The settings of DIV1D equal those in [9]. The parameters that represent physics are set as follows: Connection length $L = 5$ m; angle between magnetic field and target $\sin(\theta) = 0.06$; effective cross-field heat flux expansion $\varepsilon_\mathrm{f} = 2.3$; neutral cross-field transport $\tau_\mathrm{n} = 3$ $\mu$s.

Two scenarios are explored for the data-driven surrogates. We primarily consider the setting of ramps in the upstream density, and explore a parameter space for which DIV1D has been partially validated [9]. This setting should represent a realistic divertor plasma in TCV; we describe the details in section 3.1. Additionally, we consider a dataset with fast transients as time-dependent BCs. Since these settings have not been physically validated, this exploration is focused on evaluating the capabilities of neural PDE surrogates in more challenging settings that are potentially found in tokamak physics. These settings are described in section 3.2.

### 3.1. Density ramps

Density ramps are typically used in experiments to investigate the transition of the divertor plasma from attached to detached conditions [19]. To test the neural PDE surrogates' capabilities in forming a surrogate of DIV1D, we simulate ramps in the upstream plasma density with an exponential-like distribution to cover different timescales: $\dot{n}_\mathrm{u} \in \pm\{1.0, 2.5, 5.0, 10.0, 25.0, 50.0, 100.0\}10^{20}$ m$^{-3}$ s$^{-1}$.

The resulting simulations are between 4 ms and 400 ms, ramping up and down between $n_\mathrm{u} \in [1.0, 5.0]10^{19}\,\mathrm{m}^{-3}$. The neutral density external to the plasma is changed together with the upstream plasma density as $n_\mathrm{nb} = [2.3 - 1.6 n_\mathrm{u}\cdot 10^{-19} + 1.3(n_\mathrm{u}\cdot 10^{-19})^2]10^{17}$ (taken from [9]). Statically, the following boundary and internal conditions for DIV1D are varied: Upstream heat flux density $q_{\|\mathrm{u}} \in \{10, 15, 20, 25, 30\}$ MW m$^{-2}$; carbon concentration $\xi_\mathrm{C} \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$ ion/electron.

In summary, we create a dataset of $5\times5\times7\times2 = 350$ simulations. Initial conditions for each simulation are found by running DIV1D with static conditions and an initial guess until a steady state is reached. The numerical settings for DIV1D are as follows. The spatial domain is discretized using a finite difference scheme to a non-equidistant grid of 500 cells, with cells becoming smaller the closer they are to the target. The resulting ODE system is evolved with a timestep size of 0.001 ms using the DVODE_F90 solver [35, 36], which internally uses a variable number of timesteps (up to 100 000) for each 0.001 ms. For more details on the numerical implementation see [9].

For the NN surrogate, we use solutions on a much coarser grid than DIV1D uses internally: We linearly interpolate the cells to an equidistant grid with $Nx = 100$ points ($\mathrm{d}x \approx 0.05$ m), and use timesteps of $\mathrm{d}t = 0.1$ ms. Our simulations span between 4 ms and 400 ms, consequently, we have $Nt \in [40, 4000]$. Each data channel is standardized before being fed into the NN: The solutions are rescaled to have zero mean and unit variance for each variable (plasma density, temperature etc) over the entire dataset.

To verify that we do not lose much information by downsampling solutions we evaluate whether this discretization still represents the dominant frequencies present in the solutions. A set of DIV1D solutions with no downsampling shows that on average more than 95% of the power spectrum can be accounted for with signals below 2 kHz in the temporal axis and below three cycles per meter in the spatial axis. With the chosen discretizations sampling (more than) 5 times finer in both axes, we ensure the signal is well represented (following the Nyquist criterion [37]).

The range of values that are used with DIV1D to generate data in this paper extend outside the domain where it was shown that DIV1D provides a realistic representation of the TCV divertor plasma. However, the aim of this work is to test capabilities of machine learning methods in providing fast surrogates for flight simulator and control applications. As such the simulations contain the roll-over of the target ion flux with increasing upstream plasma density [19] and the bifurcation of the target temperature as function of upstream plasma density [18]. Ideally, these non-linear phenomena are reproduced by the machine learning surrogate of DIV1D. Both phenomena are important when the goal of the plasma exhaust is to both maintain low temperature and ion flux on the wall. Moreover, covering such phenomena greatly enhances the scope of applications for the surrogate models.

### 3.2. Fast transients

As a more challenging setting, albeit not necessarily realistic, we create a dataset generated by fast transients happening upstream. We model the transients as a spike in the upstream parallel heat flux $q_{\|\mathrm{u}}$ followed by a spike in the upstream plasma density $n_\mathrm{u}$. Note that we do not dynamically couple the external neutral density $n_\mathrm{nb}$ to $n_\mathrm{u}$ as before but leave it static. The spike in $q_{\|\mathrm{u}}$ takes 0.3 ms and is followed by a spike in $n_\mathrm{nb}$ of 1.2 ms, see figure 2. The amplitude of these spikes are chosen as a fraction of the total energy fluence and particle fluence of the incoming plasma over a period $\in$ [2.5, 20.0] ms, for a fraction $\in$ [0.05, 0.30]. For an

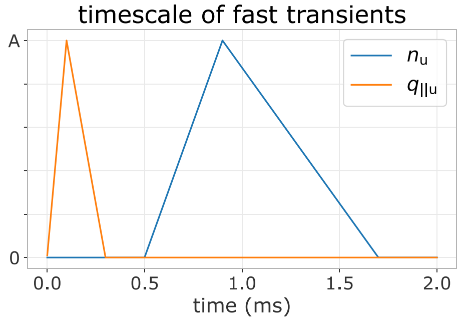


**Figure 2.** The timescale of a single transient event.

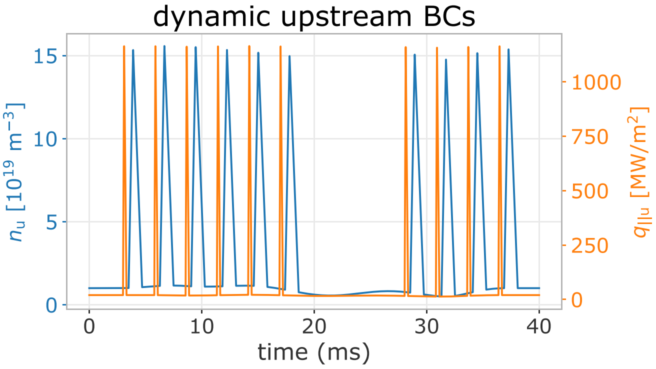


**Figure 3.** Example of dynamic boundary conditions for upstream density $n_\mathrm{u}$ and upstream parallel heat flux $q_{\|\mathrm{u}}$.

appropriate range of parameters these transients could resemble ELMs [20, 38], but the chosen parameter range is not necessarily physically valid for TCV[6].

We set the base upstream heat flux $q_{\|\mathrm{u}} \in \{10, 20, 30\}$ MW m$^{-2}$; base upstream plasma density $n_\mathrm{u} \in \{1.0, 3.0, 5.0\} 10^{19}$ m$^{-3}$; carbon concentration $\xi_\mathrm{C} \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$ ion/electron. Transients are generated with a power fraction $\in \{0.05, 0.10, 0.20, 0.30\}$ and a period of $\{2, 4, 10, 20\}$ ms, with each simulation covering 40 ms of real time. The simulations in the dataset cover the Cartesian product of these settings. Additionally, we add a set of more general simulations to the dataset. Over the duration of these simulations, we smoothly vary the background level of plasma density $n_\mathrm{u}$, the background level of heat flux $q_{\|\mathrm{u}}$, and the transients' periods and amplitudes. An example of such BCs is provided in figure 3. The complete dataset contains 1130 simulations.

The numerical settings for DIV1D are the same as before. For the resulting dataset's discretization, we again interpolate to an even grid with $Nx = 100$ points, and use a finer temporal discretization of $\mathrm{d}t = 0.01$ ms (compared to $\mathrm{d}t = 0.1$ ms for the density ramp dataset). The fast transients result in high-frequency BCs, which yield high-frequency solutions. To validate the chosen discretization, we again sample a set of solutions. More than 95% of the power spectrum can be accounted for with spatial frequencies below 3 cycles per meter (as before), and with temporal frequencies below 10 kHz. Consequently, we opt for the same $\mathrm{d}x$ as before, and refine the temporal grid by a factor of 10. With this discretization we sample at more than 5 times the highest dominant frequency in both axes, ensuring we still represent the signal properly [37].

## 4. Method

### 4.1. Method overview

The surrogate model simulates the divertor plasma following the problem formulation described in section 2. We adjust the autoregressive formulation as described in equation (3) according to developments in neural PDE surrogates. Rather than evolving one timestep at a time, bundling several timesteps together in one NN forward pass has empirically shown to improve stability and reduce computation time [12]. As such, we take both as input and output a block of time $\mathbf{t}$ rather than a single timestep $t$:

$$\begin{aligned} &\mathbf{u}^{t_{i-1},\mathbf{x}} + \mathbf{dt}_w \odot f_\theta(\mathbf{u}^{\mathbf{t}_{i-w:i},\mathbf{x}}, \mathbf{b}_\mathrm{s}, \mathbf{b}_\mathrm{d}^{\mathbf{t}_{i:i+w}}, \mathbf{c}) = \mathbf{u}^{\mathbf{t}_{i:i+w},\mathbf{x}}, \\ &\mathbf{t}_{i:i+w} = (t_i, t_{i+1}, t_{i+2}, \ldots, t_{i+w-1}), \\ &\mathbf{dt}_w = (\mathrm{d}t, 2\mathrm{d}t, 3\mathrm{d}t, \ldots, (w-1)\mathrm{d}t), \end{aligned} \tag{4}$$

where $w$ denotes the time window, the number of timesteps in each input and output block. We predict the delta of future times $t_i$ to $t_{i+w-1}$ with respect to the last known state $\mathbf{u}^{t_{i-1},\mathbf{x}}$. Intuitively, we can see equation (4) as a vectorized version of equation (3); rather than one timestep at a time, we compute $w$ timesteps in parallel. Handling time together in blocks is referred to as *temporal bundling* [12]. Since the model no longer depends on only the current state, but on the past $w$ states, we now use a short initial trajectory rather than only initial conditions for predicting full solutions. A simplified overview of the model is depicted in figure 4.

### 4.2. Model training

The objective function for optimizing parameters $\theta$ of our NN function $f_\theta$ considers the prediction error with respect to DIV1D solutions. To simplify notation we denote the LHS of equation (4) as $\mathcal{M}_\theta(\mathbf{ub}^{i-w:i}, \cdot)$, that is, model $\mathcal{M}_\theta$ predicting new states given a solution block from time $t_{i-w}$ to $t_{i-1}$ (alongside corresponding conditions, omitted for brevity). The RHS is referred to as $\mathbf{ub}^{i:i+w}$, which represents the ground truth values for the solution from time $t_i$ to $t_{i+w-1}$. The optimization target in its simplest form minimizes the one-step errors:

$$\hat{\theta} = \mathrm{argmin}_\theta\ \mathcal{L}(\mathcal{M}_\theta(\mathbf{ub}^{i-w:i}, \cdot), \mathbf{ub}^{i:i+w}), \tag{5}$$

for all simulations and timesteps in the dataset, with appropriate loss function $\mathcal{L}$. However, by only minimizing single-step errors surrogate model $\mathcal{M}_\theta$ will likely suffer from instabilities when applied iteratively. Small errors accumulate on each solver step, which will lead to the input gradually falling off the training data manifold, i.e. there is a *distribution shift*. Since the model will likely not generalize to data far out of

[6] In general it has not been assessed if DIV1D is suited to simulate the exact physics of ELMs [9].

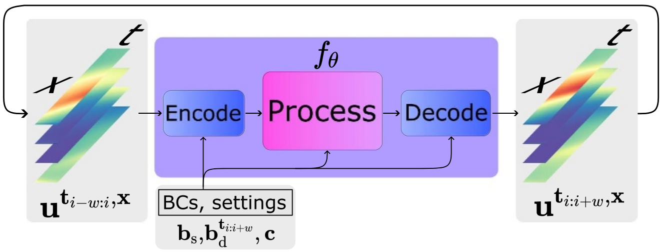

$f_\theta$
Encode
Process
Decode
$t$
$x$
$\mathbf{u}^{\mathbf{t}_{i-w:i},\mathbf{x}}$
$\mathbf{u}^{\mathbf{t}_{i:i+w},\mathbf{x}}$
BCs, settings
$\mathbf{b}_\mathrm{s},\mathbf{b}_\mathrm{d}^{\mathbf{t}_{i:i+w}},\mathbf{c}$

**Figure 4.** Overview of the method. NN function $f_\theta$ takes as input the previous subtrajectory alongside new boundary and internal conditions. Its output is the next subtrajectory, which is inserted as input for the next step. Full solutions are computed by iterating over this procedure.

its training distribution, the prediction quality will suffer. To combat this issue we also optimize with noisy model predictions as inputs, rather than only using clean DIV1D solution blocks. Consequently, the model can learn to correct its own error to stay on the data manifold. This method is referred to as the *pushforward trick* [12]. We first apply the model $n$ times to gather perturbed prediction $\widetilde{\mathbf{ub}}^{i-w:i}$, and use this noisy prediction as input. For example, for $n=2$, we get the following optimization target:

$$\begin{aligned}\widetilde{\mathbf{ub}}^{i-w:i} &= \mathcal{M}_\theta(\mathcal{M}_\theta(\mathbf{ub}^{i-3w:i-2w},\cdot),\cdot),\\ \hat{\theta} &= \mathrm{argmin}_\theta\ \mathcal{L}(\mathcal{M}_\theta(\widetilde{\mathbf{ub}}^{i-w:i},\cdot),\mathbf{ub}^{i:i+w}).\end{aligned} \tag{6}$$

Parameters $\theta$ are optimized with mini-batches of data using standard gradient-based optimization techniques. Note the separation between computing $\widetilde{\mathbf{ub}}^{i-w:i}$ and the loss calculation: The parameter gradients are only computed with respect to the final prediction step.

### 4.3. Model architectures

The formulation of the surrogate model $\mathcal{M}_\theta$ is based around $f_\theta$, a function approximation using an NN with parameters $\theta$. The architecture of this NN, which defines the function space of $f_\theta$, is key to finding a good model. Ideally, the architecture captures properties, for example, translational symmetries in the spatial domain, that fit with DIV1D solutions. However, the optimization procedure (model training) depends on this form, but this relation can be highly unpredictable: Selecting one architecture *a priori* is not sufficient. Consequently, determining the best architecture is primarily an empirical effort. In this work, we consider a representative set of architectures spanning various NN types, that show some of the best results in the field of neural PDE surrogate modeling.

As a whole, the architecture follows the encode-process-decode structure as often used in neural PDE surrogates [12, 15, 16]. The *encoder* takes an observed signal and produces an abstract representation (encoding). This representation is then evolved into an abstract representation of the next state by the *processor*. The *decoder* maps this new abstract representation back into the observed space, delivering the model output. Throughout this process the spatial geometry of the solution is maintained. For each of the components we evaluate several architectures, see figure 5 for an overview. We first summarize the encoder and decoder architectures, and subsequently describe the considered processors.

The encoder and decoder map between the observed space and the abstract space. For the encoder, input block $\mathbf{ub}^{i-w:i} \in \mathbb{R}^{Nx\times w\times 4}$ is flattened to a signal $\in \mathbb{R}^{Nx\times 4w}$, a 1D grid of $Nx$ points with $4w$ channels. We evaluate two encoder architectures, a linear convolution over the spatial domain and pointwise non-linear transformations [12]. Both map the input grid to input hidden state $h_\mathrm{in} \in \mathbb{R}^{Nx\times d}$. The decoder maps the output hidden state $h_\mathrm{out} \in \mathbb{R}^{Nx\times d}$ to model prediction $\widetilde{\mathbf{ub}}^{i:i+w} \in \mathbb{R}^{Nx\times w\times 4}$. As architectures we consider a linear convolution over the spatial domain (mapping to $4w$ channels, which are reshaped to the 4 variables over $w$ timesteps), and non-linear convolutions over hidden channels mapping to the time axis [12].

The considered processor architectures are visualized in figure 6 (see appendix A for more detailed illustrations). We categorize them according to their main underlying inductive bias (convolution-based, message passing-based and self attention-based), and summarize them in subsequent paragraphs. We implement and extend all methods in a common framework to investigate which methods work best for approximating DIV1D dynamics.

**Convolution-based networks**. Methods based on convolutional layers [39] involve learning filters/kernels that slide over a grid-based representation, making the learned transformation equivariant to shifts in its input domain. These kernels locally detect features on the grid, and by stacking these convolutional layers larger-scale behavior can be modeled. Since the kernels necessarily depend on the relative positions of grid cells, convolutional layers are tied to the grid discretization used in training.

An architecture showing much success with PDE modeling using convolutional layers is the *Dilated Residual Network* [16], abbreviated as **DRN**. Dilated convolutions [40] transform grid points not with their direct neighbors but with cells more than 1 step away, an example for 2 steps is given in figure 6(*a*). Stacking dilated convolutions with varying dilation rates allows for communication on large spatial scales while preserving local structure. These layers are implemented as a residual network [41]: For hidden representation $h_k$ as output of the $k$th layer $l_k$, rather than transforming as $h_k = l_k(h_{k-1})$, we learn the residual $h_k = h_{k-1} + l_k(h_{k-1})$.

Another architecture primarily making use of convolutional layers is the **UNet** architecture [42]. Here, the representation is first downsampled and then again upsampled in the spatial domain, whilst connecting representations of the same resolution in the down- and upsample pass; see figure 6(*b*) for an illustration. Intuitively, a UNet transforms the state on multiple spatial scales, resembling multigrid approaches. Strided convolutions are used to downsample the grid: Only one out of every $s$ grid points is processed to collect the next grid, for stride $s > 1$. Transposed convolutions are used to upsample the grid. Normalization schemes, residual connections and self-attention are also often used in modern UNet implementations [14, 43, 44]; we describe self-attention in more detail

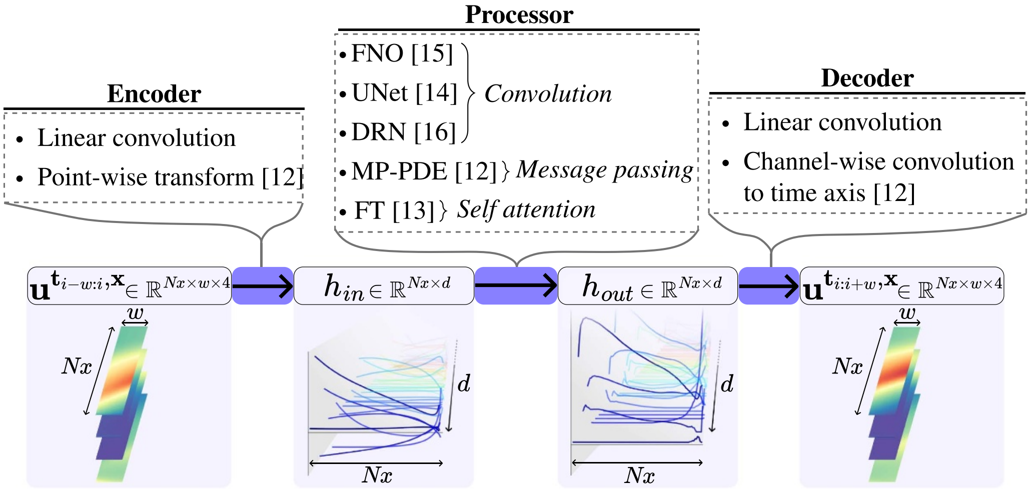
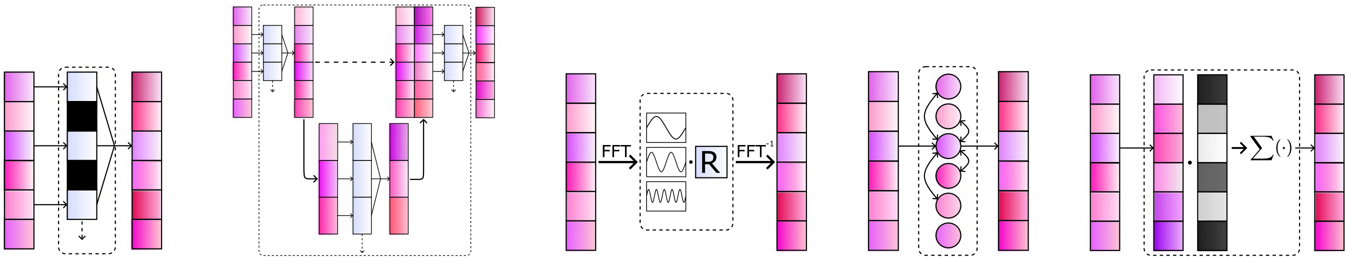

Processor
• FNO [15]
• UNet [14]
• DRN [16]
Convolution
• MP-PDE [12] Message passing
• FT [13] Self attention
Encoder
• Linear convolution
• Point-wise transform [12]
Decoder
• Linear convolution
• Channel-wise convolution to time axis [12]
$\mathbf{u}^{\mathbf{t}_{i-w:i},\mathbf{x}} \in \mathbb{R}^{Nx\times w\times 4}$
$h_{in} \in \mathbb{R}^{Nx\times d}$
$h_{out} \in \mathbb{R}^{Nx\times d}$
$\mathbf{u}^{\mathbf{t}_{i:i+w},\mathbf{x}} \in \mathbb{R}^{Nx\times w\times 4}$
w
Nx
d
Nx
d
Nx
w
Nx

**Figure 5.** Overview of the encode-process-decode framework. The encoder lifts signals from input time block $\mathbf{ub}^{i-w:i}$ to abstract representation $h_{\text{in}}$. This representation is processed (evolved) into the representation of the next state, $h_{\text{out}}$. The decoder maps this signal back to the observed space as $\mathbf{ub}^{i:i+w}$. For each component we evaluate several architectures.



(a) **DRN** (b) **UNet** (c) **FNO** (d) **MP-PDE** (e) **FT**

**Figure 6.** Simplified illustrations of different methods for transforming the hidden representation's spatial grid (vertical cells), used in the model mentioned in the caption. All methods besides the UNet are stacked sequentially to form a deep network; UNet depicts a simplified overview of the entire network architecture. Best viewed zoomed in.

below. We evaluate modern implementations of the UNet architecture as they have shown state-of-the-art performance in PDE modeling tasks [14].

Orthogonal to the aforementioned approaches, the *Fourier Neural Operator* [15], abbreviated as **FNO**, aims to learn a convolution operator in Fourier space. Rather than explicitely parameterizing convolution kernels, the spatial domain is transformed to the frequency domain by the fast Fourier transform (FFT) [45]. In this frequency representation of our hidden dimensions, a truncated $m$ spectral coefficients are multiplied by a learned weight matrix. The result is transformed back by the inverse FFT, and it is summed to a point-wise transformation of the input grid. Transforming $h_{k-1}$ to $h_k$ with such an FNO layer can be formulated as follows:

$$h_k = \sigma\big(\text{FFT}^{-1}(\mathbf{R}_k\text{FFT}(h_{k-1})) + \mathbf{W}_k h_{k-1}\big), \qquad (7)$$

for learned weight matrices $\mathbf{R}_k \in \mathbb{R}^{d\times d\times m}$ and $\mathbf{W}_k \in \mathbb{R}^{d\times d}$ ($d$ hidden dimensions; $m$ Fourier modes), and non-linear activation function $\sigma$. A simplified illustration of the principle behind the FNO is given in figure 6($c$). One reason for the FNO's power stems from the combination of linear, global integral operators and non-linear, local activation functions. A significant benefit of this formulation is the invariance to the spatial discretization: There is no direct dependence on the grid size as the hidden representation is transformed in the frequency domain (and point wise).

**Message passing-based networks**. By representing data as a graph consisting of nodes and edges, message passing NNs [46] transform a representation by updating individual nodes using a function of the node and its neighbors. Generally, a message passing step updates node $x^i_k$, where $i$ denotes the node index and $k$ the layer index, with the following formulation:

$$x^i_k = \gamma_k\Big(x^i_{k-1}, \cup_{j\in\mathcal{N}(i)}\phi_k(x^i_{k-1}, x^j_{k-1}, e^{j,i})\Big), \qquad (8)$$

where $\phi$ denotes the *edge transfer function*, $\cup$ denotes the *aggregation function*, $\gamma$ denotes the *node update function*, $\mathcal{N}(i)$ denotes the indices of the neighbors of $x^i$, and $e^{j,i}$ denotes the data associated with the edge between nodes $x^i$ and $x^j$. Functions $\phi$ and $\gamma$ are parameterized by small NNs, whereas aggregation $\cup$ is usually a simple function such as the mean or sum of the edge embeddings. As long as $\cup$ is invariant to the order of the edges, the network as a whole will be equivariant with respect to permutations of the input graph.

In the case of PDE modeling we represent the grid as a geometric graph, i.e. nodes are defined by the features on the gridpoint and the grid coordinates (consequently, the network is no longer equivariant to permutations of node features). Nodes are connected according to relative distances on the grid, for example by adding edges between points that lie within 2 cells from each other. We use graph neural networks (GNNs) to benefit from their expressive power: One can consider a message passing step as a generalization of a convolution. Additionally, since GNNs only operate on relations between nodes through their edges, GNNs are in principle capable of using arbitrary spatial discretizations. A notable implementation of GNNs for PDE modeling we evaluate is the *Message Passing PDE* solver, abbreviated as **MP-PDE** [12], which implements message passing according to differences in features and positions of nodes. A simplified illustration of message passing is provided in figure 6(*d*).

**Self attention-based networks**. The self attention mechanism, most prominently used in the transformer architecture [44], operates on sequences by transforming each element according to all other elements in the sequence. A dynamically weighted attention score is computed for all pairs of elements in the sequence, which is used in tandem with another transformation of the input sequence to compute the output sequence. Self attention is equivariant to permutations in the sequence as no information regarding the position of elements is used in this computation. One strength of self attention is its expressiveness, as it merges information over the entire sequence with dynamically computed weightings.

In more detail, elements are transformed by three learned matrices: Query matrix $\mathbf{W_Q} \in \mathbb{R}^{d\times d}$, key matrix $\mathbf{W_K} \in \mathbb{R}^{d\times d}$ and value matrix $\mathbf{W_V} \in \mathbb{R}^{d\times d}$, for elements with $d$ dimensions. We denote the transformed sequences as $\mathbf{Q} \in \mathbb{R}^{n\times d}$, $\mathbf{K} \in \mathbb{R}^{n\times d}$ and $\mathbf{V} \in \mathbb{R}^{n\times d}$, for sequences of length $n$. The attention score is computed as the softmax over the product of sequences $\mathbf{Q}$ and $\mathbf{K}^\top$ normalized by the number of hidden dimensions. This score is then multiplied by $\mathbf{V}$ to generate the final output[7]:

$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}. \tag{9}$$

An alternative interpretation of the self attention mechanism is provided by [13], where rather than considering rows of embedded sequences $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ as point-wise feature embeddings, we can consider the columns as vector representations of learned basis functions of the representation. Motivated by this interpretation they provide alternative formulations of the self-attention mechanism for PDE modeling, of which we use the *Transformer with Fourier-type Attention*. We refer to this model as the Fourier Transformer, abbreviated as **FT**. Fourier-type attention is formulated as follows:

$$\mathrm{Attention}_{\mathrm{FT}}(\mathbf{Q},\mathbf{K},\mathbf{V}) = (\widetilde{\mathbf{Q}}\widetilde{\mathbf{K}}^\top)\mathbf{V}/n, \tag{10}$$

where $\widetilde{\diamond}$ denotes layer normalization [47] and $n$ the number of elements in the sequence. Since our sequence represents the spatial grid, $n = Nx$. This attention formulation alongside residual connections and a point-wise NN form the basis of the (Fourier) Transformer. Positional information is concatenated with the element features to let the model use positional relations of grid points; consequently, it is no longer equivariant to permutations of the input sequence. Additionally, since the positional information is used only as input feature, a transformer is in principle capable of working with arbitrary spatial discretizations. A simplified illustration of self-attention is provided in figure 6(*e*). We also evaluate the use of FT layers followed by FNO layers (as proposed in the FT paper [13]), denoted as **FT-FNO**.

### *4.4. Adaptations for DIV1D data*

Besides implementing various existing methods in a unified framework for training autoregressive encode-process-decode models with temporal bundling and the pushforward trick, we also make adaptations tailored towards the DIV1D data. Specifically, we address two properties: A wide variation in simulation length $Nt$ (as we model density ramp dynamics spanning multiple timescales) and the dependence of solutions on (time-varying) conditions. To address the former, we propose an adapted strategy for sampling batches while training. To address the latter, we describe a simple conditioning method that we implement across all architectures described in section 4.3.

**Sampling with scheduled unrolling**. As described in section 3.1, simulations in the density ramp dataset range from 40 to 4000 timesteps. We found that using many unrolling steps with the pushforward trick (section 4.2; [12]) is key to training models that are stable over long timeframes. However, since training is done using minibatches of randomly sampled datapoints (to exploit the parallel computation of GPUs), simulations with large differences in length are often batched together. Due to these differences in length, long unrollings during training require tricks to deal with simulations shorter than the unrolling window, such as padding and adapting the model input. These tricks come at a cost: For example, when processing a simulation of length 40 in a batch where we unroll 10 times with $w = 20$, the forward computation is done on 200 timesteps, whereas only 20 timesteps can be used for the pushforward loss computation. The redundant computations amount to a significant part of the total cost, making training unnecessarily long and expensive[8].

To combat this issue we propose sampling batches by first sampling an unrolling window and then sampling items that fit in this window. Sample probabilities are adjusted such that all items are sampled uniformly in expectation, as we do not want to bias the learning algorithm towards dynamics occurring in longer simulations. We sample unrolling length $t$ from

[7] For simplicity we describe the case of only a single attention head in so-called multi-head attention. In multi-head attention, feature dimensions $d$ of each sequence element are split over multiple attention heads, i.e. each self-attention computation transforms a subset of the data's dimensions.

[8] While the gradient computation, which on itself is most expensive, is done only once at the final prediction step, we found that when unrolling for many steps the majority of time in each optimization step is spend in the unrolling procedure.

a chosen distribution $p(t)$ and sample items $x$ from distribution $p(x|t)$ which is constrained as follows:

$$\mathbb{E}_{t\sim p(t)}\left[p(x_i|t)\right] = \frac{1}{N}, 0 \leqslant i < N, \tag{11}$$

where $N$ denotes the total number of simulations in the dataset. We refer to appendix B for details on the computation of $p(x|t)$, an evaluation of multiple unrolling distributions, and a comparison with baselines.

**Incorporating (time-varying) conditions**. Our function approximation $f_\theta$, and by extension surrogate model $\mathcal{M}_\theta$, not only depends on the previous state of the system but also on (time-varying) BCs and internal conditions. These conditions are inserted into the model's internal representations as follows. We process conditions $(\mathbf{b}_\mathrm{s}, \mathbf{b}_\mathrm{d}^{\mathbf{t}_{i:i+w}}, \mathbf{c})$ to a conditioning vector **cond**, and concatenate this vector throughout the internal representations. In more detail, **cond** is created by first processing the time-varying conditions $\mathbf{b}_\mathrm{d}^{\mathbf{t}_{i:i+w}}$ with a small NN and concatenating this output with $\mathbf{b}_\mathrm{s}$ and $\mathbf{c}$. This vector is repeated for each grid point, making our conditioning vector **cond** $\in \mathbb{R}^{Nx\times dc}$ for $dc$ conditioning features. To insert this information into the model, we implement the same strategy for all architectures: **cond** is concatenated feature-wise to all spatial hidden representations $h \in \mathbb{R}^{Nx\times d}$, such that the model learns mappings from $\mathbb{R}^{Nx\times(d+dc)} \to \mathbb{R}^{Nx\times d}$, i.e. the model can make use of the simulation conditions for each internal transformation. Additionally, we correct the model output at each step: For the plasma density, plasma temperature and neutral density we clamp outputs to a minimum value of 0.1, and we set the upstream plasma density values to the time-varying BCs.

## 5. Experiments and results

In this section we evaluate all methods and adaptations with the purpose of constructing a fast and accurate high-fidelity surrogate model of divertor plasmas, based on data from DIV1D. We primarily focus on the density ramp dataset, as this data represents a realistic divertor plasma; the objective of the fast transient dataset is to provide an exploratory outlook towards modeling higher frequency dynamics.

We start with a short summary of the training procedure and settings in section 5.1. In sections 5.2 and 5.3 we evaluate all methods and propose the configuration for surrogate model DIV1D-NN using the density ramp dataset. In section 5.4 we evaluate DIV1D-NN in more depth through a set of case studies using the density ramp data, and also evaluate its ability to scale to more difficult datasets and dynamics with the fast transient data. Finally, in sections 5.5 and 5.6, we investigate properties of the surrogate with respect to the datasets, focusing on the number of simulations in the training data and the surrogate's inter- and extrapolation capabilities.

### 5.1. Method recap and hyperparameters

The two datasets are split as follows. For the density ramp dataset (350 simulations), we use 300 for training, 25 for validation (model selection) and 25 for testing (final results). The split is randomly sampled for the most part—the only intervention is that we ensure the test set contains some simulations with identical parameters both as a density ramp up and ramp down. The split is kept fixed throughout the experiments unless stated otherwise. For the fast transient dataset (1130 simulations), we use 904 for training, 113 for validation and 113 for testing. The test split is selected to contain all 60 simulations with transients where the average energy fluence is between 15 kJ m$^{-2}$ and 20 kJ m$^{-2}$, along with 53 randomly sampled simulations (covering 10% of the dataset in total). The remaining simulations are randomly split between the train and validation set.

For training we use a batch size of 16. The procedure consists of first sampling a batch of simulations and the unrolling time (section 4.4). For all simulations in the batch we pick a random starting point, unroll the model predictions, compute and backpropagate the loss and update the model parameters (section 4.2). This process is repeated for $\lceil\frac{300}{16}\rceil = 19$ batches per epoch for the density ramp dataset and $\lceil\frac{904}{16}\rceil = 57$ batches per epoch for the transients dataset. That is, one epoch is one full pass over the dataset, and we repeat this procedure for 20 000 epochs for both datasets. Following standard practice, we select the model parameters for which the validation set error is minimal, and report results on the test set.

For the density ramp dataset, we use the root of the squared error (i.e. the $L_2$ distance) as loss function:

$$\mathcal{L}(\mathbf{ub}, \tilde{\mathbf{ub}}) = \sqrt{\textstyle\sum_{i=1}^{n} |\mathbf{ub} - \tilde{\mathbf{ub}}|^2}, \tag{12}$$

for all $n$ points in the batch, where **ub** denotes a time block of DIV1D reference solutions and $\tilde{\mathbf{ub}}$ denotes the corresponding model predictions. For the fast transient dataset, we average the squared error and absolute error and take the square root:

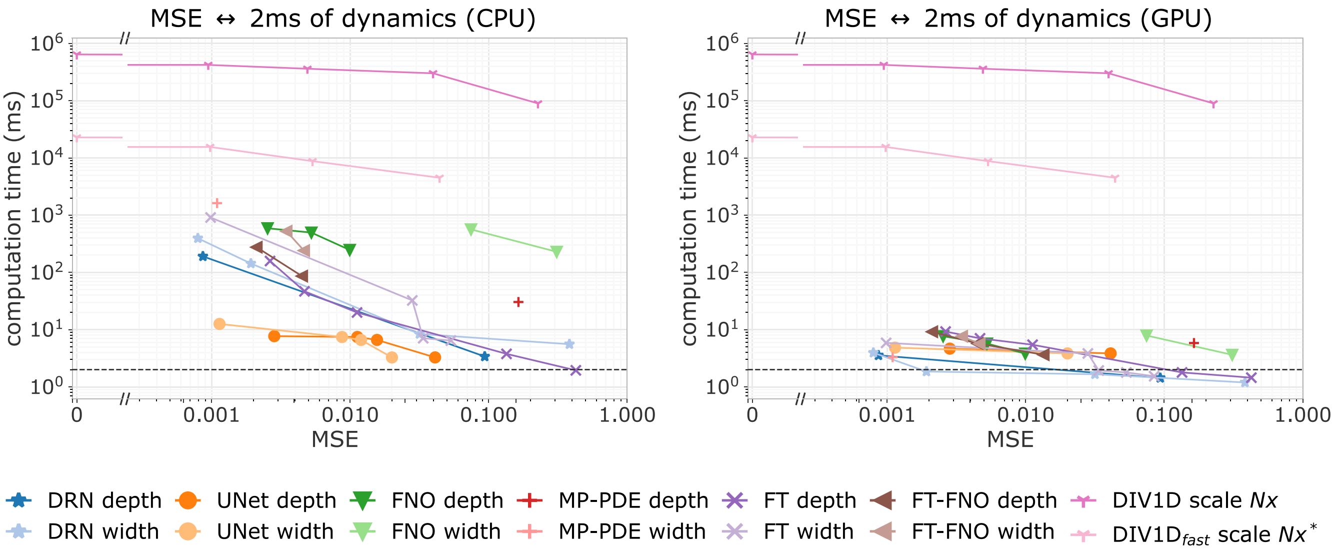
$$\mathcal{L}(\mathbf{ub}, \tilde{\mathbf{ub}}) = \sqrt{\textstyle\sum_{i=1}^{n} \left(0.5 \cdot |\mathbf{ub} - \tilde{\mathbf{ub}}| + 0.5 \cdot |\mathbf{ub} - \tilde{\mathbf{ub}}|^2\right)}, \tag{13}$$

with the same variables as before. We use both terms as we empirically found that the absolute error term helped stabilize training in the presence of large spikes in the solutions. We use the Adam optimizer [48] with an initial learning rate of $10^{-4}$ and decay the learning rate by 0.4 at epochs 500, 2500, 5000 and 7500.

All models compute the solution in blocks of 2 ms, which corresponds to $w = 20$ timesteps for the density ramp dataset (where d$t = 0.1$ ms) and $w = 200$ timesteps for the fast transient dataset (where d$t = 0.01$ ms). All models are implemented using PyTorch [49], and are trained and evaluated using an NVIDIA A100 40GB GPU and Intel Xeon Platinum 8360Y CPU unless stated otherwise.

### 5.2. The trade-off between error and computation time

We investigate different architectures for the model alongside different hyperparameters for each architecture, with the aim of finding a fast and accurate surrogate; the selected configuration is described in section 5.3. These evaluations are done



(a) AMD Rome 7H12 CPU, 1 core

(b) NVIDIA A100 GPU

**Figure 7.** Work-precision diagrams of the MSE for full solution rollouts versus the computation time of 1 block (2 ms of plasma simulation). We compare all neural PDE surrogates with default DIV1D: Neural PDE surrogates are scaled by their architecture, DIV1D is scaled by decreasing the size of the computational grid. We benchmark using (*a*) equal hardware and (*b*) GPUs, if applicable. The dashed line represents the real-time barrier. Note the disconnect in the *x*-axis: The computation cost of reference solutions is also visualized, but since by definition they have an MSE of 0, they cannot be plotted on a log-scale. $^*$For DIV1D$_{\text{fast}}$, the MSE is calculated w.r.t. its own solutions at $Nx = 500$ rather than the DIV1D reference solutions.

using the density ramp dataset, aiming to simulate density ramps in a realistic TCV divertor plasma.

For the encoder and decoder we evaluate two architectures, and for the processors we evaluate the DRN, UNet, FNO, MP-PDE, FT, and FT-FNO (section 4.3). Since the emphasis of the surrogate model lies on fast computation, we evaluate configurations for a range of inference speeds. For one block (2 ms of real time) we search configurations with a computation time of $\approx\{0.5, 1, 2, 4, 6, 8, 10\}$ ms. We search for parameters aiming at 'wide' and 'deep' networks, that is, putting the emphasis on a network with many features per block, or on stacking many blocks. For each processor we define a maximum width and maximum depth configuration and iteratively reduce it until it reaches the desired computation time. For details on these configurations and the identified settings we refer to appendix C.

We compare predictions on simulations from the test set through work-precision diagrams. Quality is measured through the mean squared error (MSE) of the solutions, starting after the input time block of *w* timesteps:

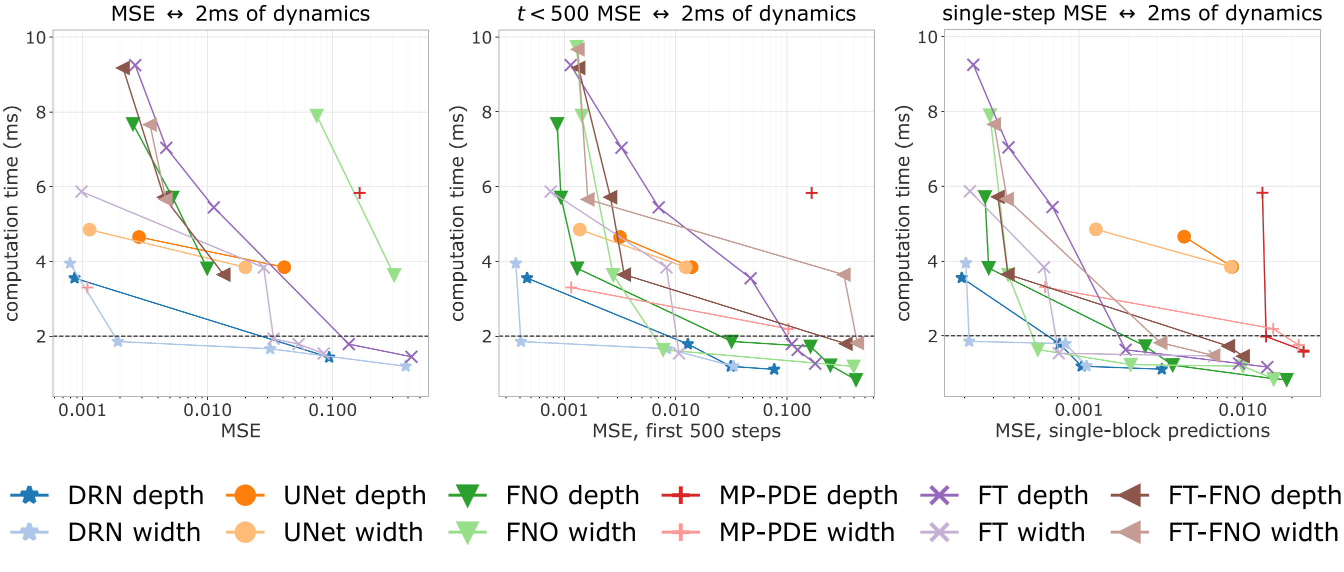
$$\text{MSE}\left(\mathbf{u}^{\mathbf{t},\mathbf{x}},\tilde{\mathbf{u}}^{\mathbf{t},\mathbf{x}}\right) = \frac{1}{(Nt-w)\cdot Nx}\sum_{t=t_w}^{t_{Nt}}\sum_{x=x_0}^{x_{Nx}}\left(\mathbf{u}^{t,x}-\tilde{\mathbf{u}}^{t,x}\right)^2, \tag{14}$$

for the discretized DIV1D reference solution $\mathbf{u}$ and the corresponding prediction $\tilde{\mathbf{u}}$. We measure with variable-wise standardized solutions so all variables are on the same scale. As a reference, since the test set will have a mean of approximately 0 and a variance of approximately 1, a naïve baseline of predicting all zeros will result in an MSE of approximately 1. Speed is measured as the time to compute 2 ms worth of solutions, which corresponds to 1 output block. In other words, we compute the latency of the model with a batch size of 1 (the total number of solutions per second can be increased by using bigger batch sizes, which better exploits the parallel computation of GPUs).

In the work-precision diagrams (figures 7(*a*) and 8), each processor architecture and parameter-search strategy (deep or wide networks) is shown as one line: Each point represents a different configuration plotted at its speed and error. We plot the Pareto front of each such category to keep the plots readable, and omit all points with an MSE of more than 0.5. For a complete overview of experimental results we refer to appendix C.

**Comparison with DIV1D**. We start by comparing full-length simulations. To place the surrogates in context, we compare them to DIV1D's speed-accuracy trade-off when coarsening its spatial grid. To evaluate the computation time of the neural PDE surrogates with a reasonably optimized code, an effort was made to accelerate the DIV1D code. By considering that neutrals have finite energy in the calculation of charge exchange energy losses [9, equation 11(a)], the original DIV1D implementation is accelerated considerably. We denote this faster version as DIV1D$_{\text{fast}}$. While the solutions are qualitatively similar this deviation in the computation results in notable differences on the gridpoint level. Therefore, we compute all DIV1D/DIV1D$_{\text{fast}}$ errors w.r.t. their own best-quality solutions at $Nx = 500$.

In figure 7(*a*) the comparisons are plotted for equal hardware (AMD Rome 7H12 CPU, 1 core). The left-most points



(a) Full simulations (b) First 500 steps (c) Single-block predictions

**Figure 8.** Work-precision diagrams for the neural PDE surrogates for: (*a*) full solution rollouts; (*b*) solutions of 500 steps; and (*c*) predicting 20 timesteps (one time block) from arbitrary starting points. Computation time is measured on an NVIDIA A100 GPU. The DRN dominates the Pareto front in most cases, although this advantage is smaller for shorter time windows.

for DIV1D and DIV1D$_{\text{fast}}$ correspond to the reference solutions (hence 0 error and the 'gap' in the log-scale *x*-axis). On equal hardware, neural PDE surrogates already show significant speed-ups compared to DIV1D while keeping high accuracy: The surrogates' best accuracy is comparable to the first step down for DIV1D and DIV1D$_{\text{fast}}$, which corresponds to running them using 450 grid points, down from 500 grid points for the reference solutions.

Since NN methods benefit from parallel computation and the use of dedicated accelerators such as GPUs, we evaluate the surrogates using a fast GPU; these results are plotted in figure 7(*b*). While this comparison is not even, it is highly non-trivial to exploit this hardware with existing CPU-based numerical codes such as DIV1D. As such, we primarily focus on the fastest possible computation times. Figure 7(*b*) shows that the neural PDE surrogates can generate accurate solutions several orders of magnitude faster.

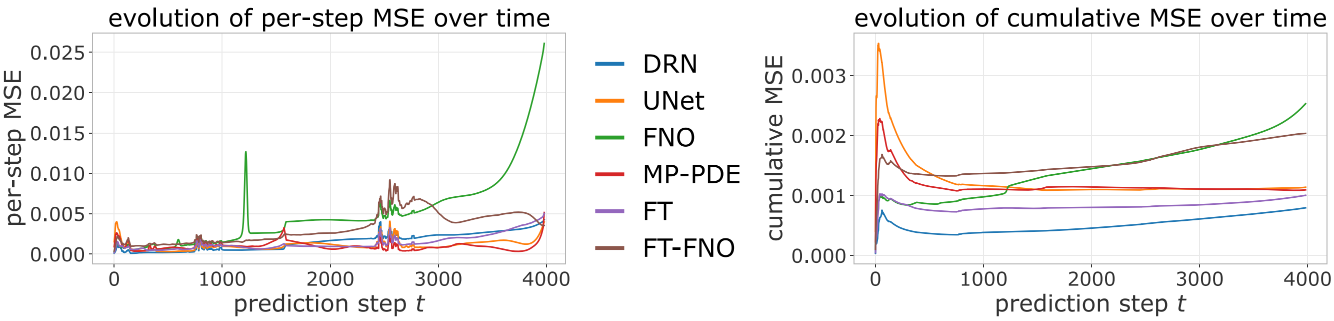
**Comparison of neural PDE surrogates**. To compare surrogate methods more precisely we zoom in on the surrogates only, see figure 8(*a*) for full-simulation errors for all methods on the GPU. The DRN shows strong performance, both for computing maximum accuracy solutions at any cost and for computing accurate solutions fast. Notably, the UNet's speed-accuracy trade-off is very favorable when using a single CPU core (figure 7(*a*)), but this advantage fades when evaluating on the GPU. Likely, the relatively deep and complex architecture of a UNet is not as favorable for the parallelized GPU computations.

Since the solutions can cover many timesteps–the longest simulations being 4000 timesteps–error accumulation becomes a major factor. Methods could be viable for short-term predictions but show bad performance over long rollouts. To evaluate the latter, we check the error for the MSE for the first 500 steps (50 ms) of all simulations, and errors of single block predictions (2 ms) given arbitrary starting points in the simulation. These are plotted in figures 8(*b*) and (*c*), respectively. The DRN still performs best, but the lead is less pronounced.

To further evaluate the influence of error accumulation we plot the error of simulations over time. For each processor architecture we select the model with the lowest MSE and plot this error as a function of time. In figure 9(*a*) the error is plotted at each individual timestep, whereas figure 9(*b*) displays the cumulative MSE evolving over time (with the final timestep being the total MSE as used in figure 8(*a*)). While most models have a similar distribution over time, the FNO stands out: At short timeframes its error is comparable to the DRN and FT, but it accumulates more error over time. The opposite holds for the MP-PDE and UNet, which show more stability w.r.t. time.

### *5.3. Selecting the NN surrogate architecture for DIV1D*

For the final divertor plasma surrogate model, we select the configuration we deem to have the best speed-accuracy trade-off. This model is the DRN with an inference speed of ≈1.807 ms per block and a standardized MSE of 0.001 918. We dub this configuration as *DIV1D-NN*. To push the computation time as low as possible we further optimize this implementation and compile the model with the NVIDIA TensorRT SDK [50] for fast inference. These optimizations result in an inference speed of ≈0.6221 ms per block, about 3 times as fast as the simulated plasma dynamics of 2 ms. Figure 11 depicts a comparison of compute times with varying hardware and with DIV1D. For context, training DIV1D-NN (the offline cost) took just under 4 h.



(a) Per-step MSE over time

(b) Cumulative MSE over time

**Figure 9.** Plots of test simulation error as a function of time. The per-step error is displayed in (*a*), whereas (*b*) shows the cumulative error. The common spike at the start in (*b*) can be explained due to the impact of short simulations: These dynamics are faster and more challenging, and only affect the error computation for their short time span.

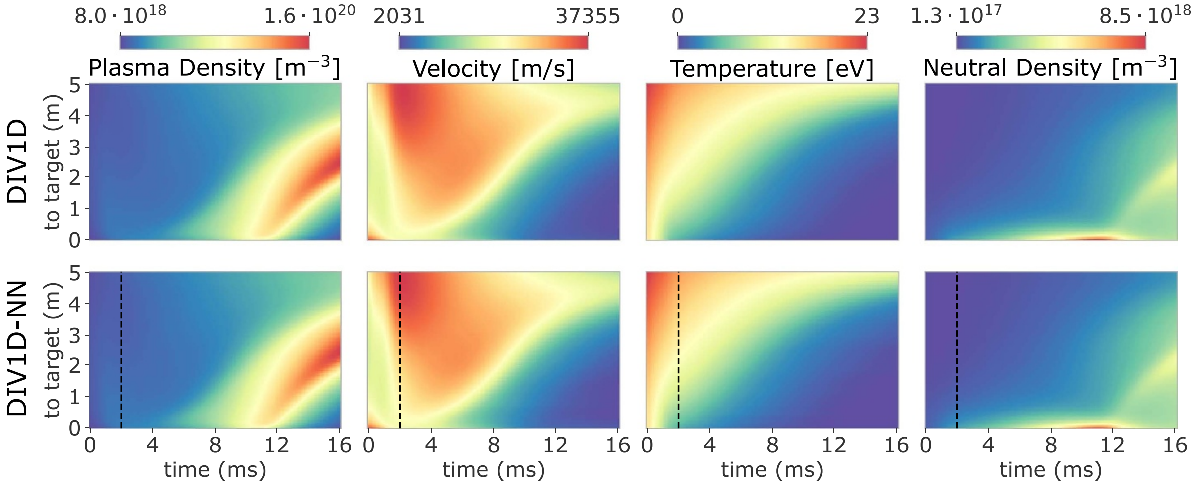


**Figure 10.** Visualization of a simulation from the test set: a ramp up of $n_\mathrm{u} = [1.0 - 5.0]10^{19}$ m$^{-3}$ over 16 ms, $q_{\|\mathrm{u}} = 15$ MW m$^{-2}$, $\xi_\mathrm{C} = 0.04$ ion/electron. The top row depicts the refence DIV1D simulation, the bottom row the DIV1D-NN simulation. The dashed line indicates the end of the first block, the input for DIV1D-NN.

As qualitative comparison we plot a DIV1D-NN simulation alongside the reference DIV1D simulation in figure 10, of dynamics induced by a ramp up. The top row depicts the reference simulation, whereas the bottom row depicts the DIV1D-NN simulation. Qualitatively, these simulations align closely.

### 5.4. Case studies: recovering properties and structures

To assess the utility of DIV1D-NN for downstream tasks we evaluate its performance on recovering a set of relevant properties and structures. In particular, we consider the ability to reconstruct two non-linear phenomena: A roll-over of the target ion flux with increasing upstream plasma density [19] and a bifurcation of the target temperature as a function of upstream plasma density [18]. These phenomena are important when aiming for divertor plasmas that maintain both a low temperature and ion flux on the target. Additionally, we evaluate the reconstruction of the approximate emission front, a useful proxy for detachment control [17]. We repeat a subset of these evaluations for fast transient behavior by retraining DIV1D-NN on the fast transients dataset and re-evaluating the results.

Key observation is that we do not explicitly train DIV1D-NN for any of these properties, but rather evaluate whether it is sufficiently accurate w.r.t. DIV1D such that the surrogate can fill various roles of the source model without building a surrogate for each role. An additional benefit relative to building individual surrogates is that we can still evaluate the full trajectories related to these predictions. If a trajectory can be identified as non-physical we can discard the prediction, making our surrogate modeling strategy less of a black box compared to methods that directly map input parameters to target quantities.

**Bifurcation in target temperature**. First, we evaluate whether DIV1D-NN accurately captures the bifurcations (hystereses) in the target temperature. That is, in certain otherwise identical conditions, the target temperature varies depending

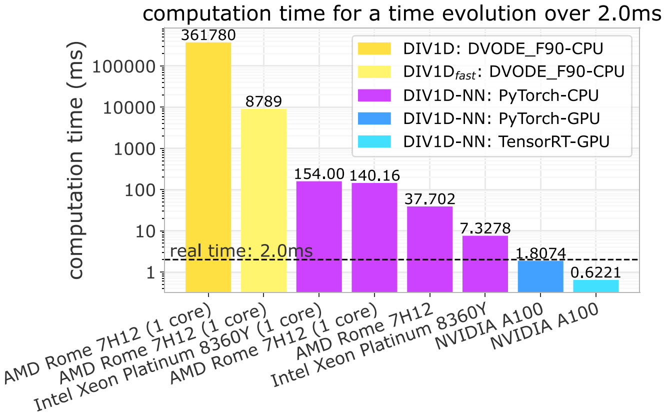


**Figure 11.** Computation times of DIV1D-NN with varying hardware and inference engines, along with DIV1D for comparison. DIV1D-NN's speed is computed as the time of one model forward pass, generating a block of 20 timesteps spanning 2 ms of dynamics. DIV1D's speed is the average time to generate 2 ms of data from the test set. Since DIV1D-NN incurs some error, we use the first setting for DIV1D and DIV1D$_{\mathrm{fast}}$ that resulted in a higher error than DIV1D-NN ($Nx = 400$ for both, third points from the left in figure 7(*a*)).

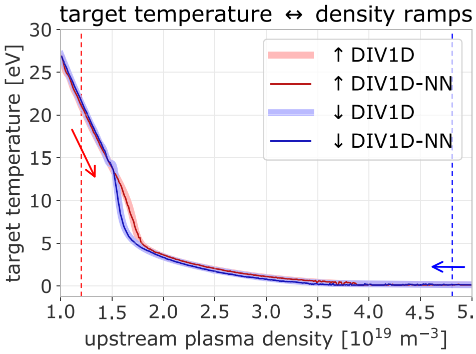


**Figure 12.** Target temperature as function of upstream plasma density, comparing DIV1D and DIV1D-NN for 40 ms ramps with $q_{\parallel \mathrm{u}} = 25$ MW m$^{-2}$ and $\xi_{\mathrm{C}} = 0.05$. The plot depicts both a ramp up and ramp down from the test set, with red indicating the ramp up and blue indicating the ramp down. DIV1D-NN matches the bifurcation captured by DIV1D well, with minor artifacts.

on whether the upstream density goes from low to high or vice versa [18]. We illustrate this bifurcation, with DIV1D-NN compared to DIV1D, in figure 12. We find a close match, with only small deviations in the DIV1D-NN simulation. Quantitatively, the standardized MSE of the target temperature is 0.001 007 over all test simulations. Rescaled to the physical scale, the average absolute error between predicted and real target temperatures is 0.1714 eV.

**Target ion flux roll-over**. Next, we consider whether DIV1D-NN accurately captures the roll-over in the ion flux [19], as well as its dependence on the ramp speed. This roll-over can be demonstrated by looking at the target neutrals, due to recombination at the target being proportional to the target ion flux. In figure 13 the neutral density at the target is plotted as function of upstream density for varying density rates. The roll-over is reconstructed accurately, with the influence of time dynamics clearly illustrated. Quantitatively, the standardized MSE of the target neutral density is 0.083 85 over all test simulations. In the physical scale, this error corresponds to an average absolute error of $2.2873 \cdot 10^{17}$ m$^{-3}$.

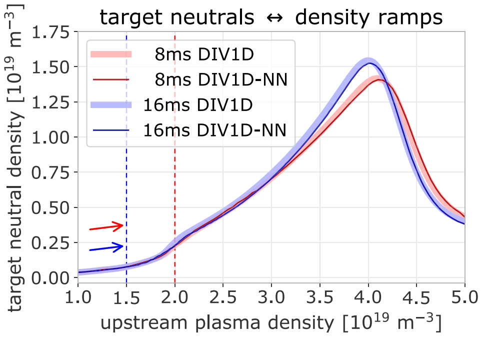


**Figure 13.** Target neutral density as function of upstream plasma density, comparing DIV1D and DIV1D-NN for an 8 ms and 16 ms ramp up with $q_{\parallel \mathrm{u}} = 30$ MW m$^{-2}$ and $\xi_{\mathrm{C}} = 0.05$. The roll-over of target ion flux results in a similar roll-over in the target neutrals; this phenomenon is captured by DIV1D, and recovered by DIV1D-NN.

**Emission front position**. As final property we consider the approximate location of the carbon impurity emission front, defined as the position along the flux tube where the plasma temperature is equal to 7 eV. The 7 eV temperature corresponds to the peak of the cooling rate function from [51] used by DIV1D. Ideally, this temperature corresponds to the temperature of the CIII impurity front position as used in [17, 52]. In reality the exact temperature of the front position depends on the plasma scenario and is hard to infer (see [53, 54] and references therein). As definition of detachment we follow [55], who define it as the moment where the target temperature is below 10 eV, but we use the 7 eV temperature as it conveniently corresponds to our definition of the carbon impurity emission front. The position is detected by scanning the generated temperature profiles, filtering them such that they are monotonically decreasing from upstream towards the target, and interpolating between gridcells to find the location at 7 eV. We scan from upstream towards the target; if no point below 7 eV is found, we assume the front to lie at the target, corresponding to an attached divertor plasma.

In figure 14 we overlay the predicted emission front on top of the estimation for a reference DIV1D simulation for a ramp up. Even over the long simulation timeframe of 160 ms, the DIV1D-NN prediction still closely aligns with DIV1D. The average absolute error of the predicted emission front location over the entire test set is 0.015 70 m. Additionally, we can use the predicted emission front location to estimate detachment in the simulated plasma: We assume detached if the temperature at the target is smaller than 7 eV, and attached otherwise. A confusion matrix for this evaluation is provided in table 1. In most cases DIV1D-NN accurately captures the state modeled by DIV1D, with an accuracy of ≈99.87% over the entire test

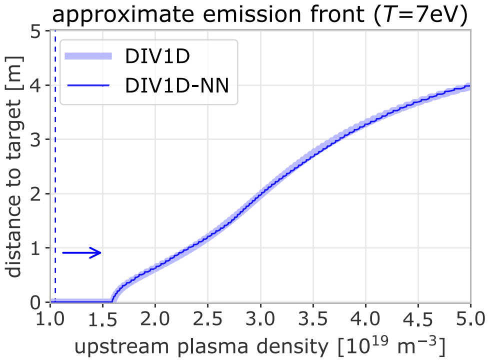


**Figure 14.** Approximate emission front as function of upstream plasma density, comparing DIV1D and DIV1D-NN for a 160 ms ramp up with $q_{\|\mathrm{u}} = 20$ MW m$^{-2}$ and $\xi_{\mathrm{C}} = 0.02$.

**Table 1.** Confusion matrix of attachment predictions, where attachment is defined as the target temperature being below 7 eV. The accuracy of predicting attachment is 99.87% over the entire test set.

| | | DIV1D | |
|---|---|---|---|
| | | Detached | Attached |
| DIV1D-NN | Detached | 9557 | 7 |
| | Attached | 9 | 2612 |

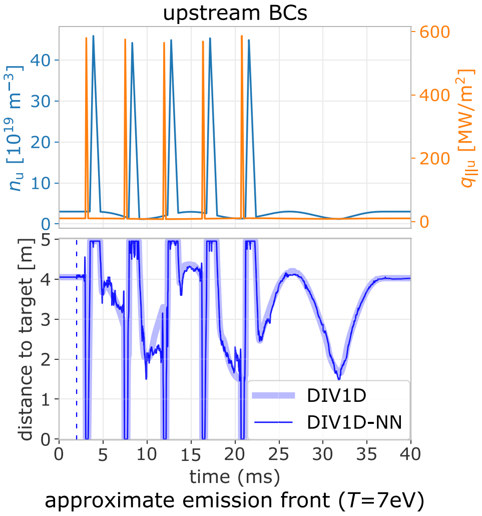


**Figure 15.** Comparing DIV1D and DIV1D-NN for tracking the emission front over a period with fast transients. The corresponding BCs, the upstream density $n_\mathrm{u}$ and upstream parallel heat flux $q_{\|\mathrm{u}}$, are depicted on top.

set (as a baseline, ≈78% accuracy could be achieved by always predicting detachment).

**Fast transients.** To investigate the capabilities of the DIV1D-NN surrogate architecture in a more challenging setting, we retrain this architecture from scratch using the fast transients dataset. These transients could resemble phenomena like ELMs, representing much smaller timescales with much larger gradients than found in the density ramp dataset; the corresponding solutions are much less smooth than those trained and tested with before. Training DIV1D-NN on the fast transient data took about 12 h of wall-clock time.

Since we predict in blocks of 2 ms, we now use window size $w = 200$ instead of $w = 20$ to account for the shift in time discretization d$t$ from 0.1 ms for the density ramp data to 0.01 ms for the fast transient data. Note that for the transient data, we always start simulations with a steady-state, i.e. all 200 timesteps in the first block are identical.

The architecture of the model's processor is identical to DIV1D-NN as used before, the change in input and output dimension is accounted for in the encoder and decoder respectively. As a result, model inference is slightly slower, taking ≈0.6788 ms per block to compute 2 ms of plasma dynamics (≈0.6221 ms before). Even though we compute at a 10 times finer temporal resolution the effect on computation time is somewhat limited, because in the processor the dimensionality was already scaled up beyond the time discretization, and this dimensionality remains unchanged between the two datasets' models. Additionally, we now directly predict solution values, rather than the delta as written in equation (4). We found that models struggled to train when predicting the delta, as the correct output values then heavily depend on whether the input block ended on top of a spike in the solution or not.

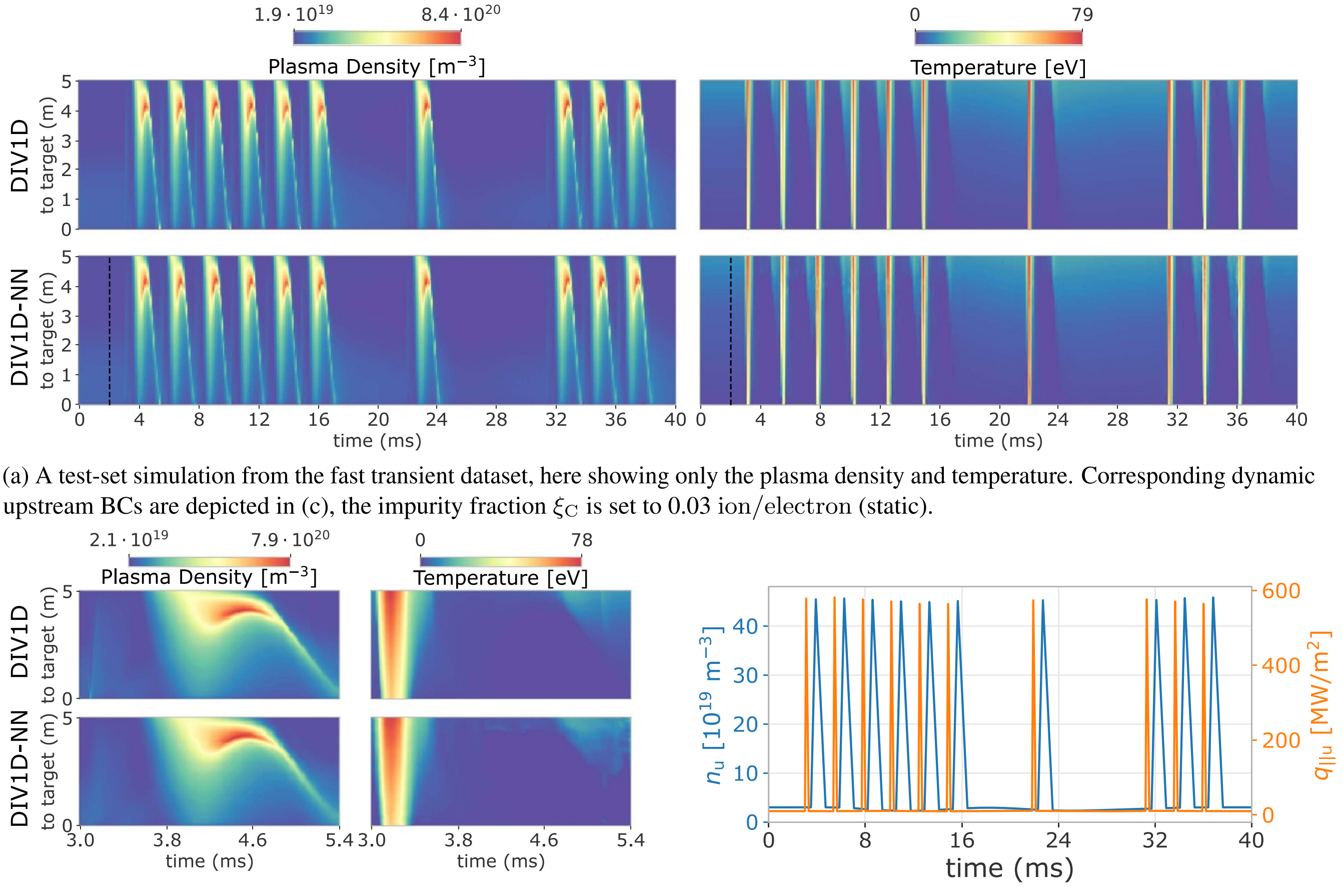
The MSE on standardized data using DIV1D-NN is 0.022 98, which is comparable to running DIV1D with just below 400 grid points, instead of the 500 grid points used for the reference solutions (see appendix D for DIV1D's scaling on the fast transient data). The NN surrogate does not scale as well as on the density ramp data, stressing the challenge for data-driven surrogates when modeling high-frequency dynamics. However, large-scale features are still recovered well, see figure 16 for an example solution.

Of interest is whether aggregate structures can be captured well in the fast transient setting. To evaluate this question, we compare the estimated emission front locations for fast transient simulations. The average absolute error of the front location is 0.053 74 m, with the detachment prediction having an accuracy of 98.61%; see appendix D for the confusion matrix. The model is still quite accurate, but there is a drop in quality compared to results on the smoother density ramp dynamics. An example is provided in figure 15. The position is tracked well on a global scale, but on the small scale, such as the regions between the spikes, there are noticeable artifacts.

### 5.5. Data efficiency

When creating a data-driven surrogate model, the quality of the resulting model is highly dependent on the size of the dataset. Generating a rich training dataset can be a severe bottleneck in the surrogate creation process[9]. To investigate the

[9] Another avenue for improving data efficiency considers identifying which simulations are most informative to the surrogate, i.e. *Active Learning* [56].

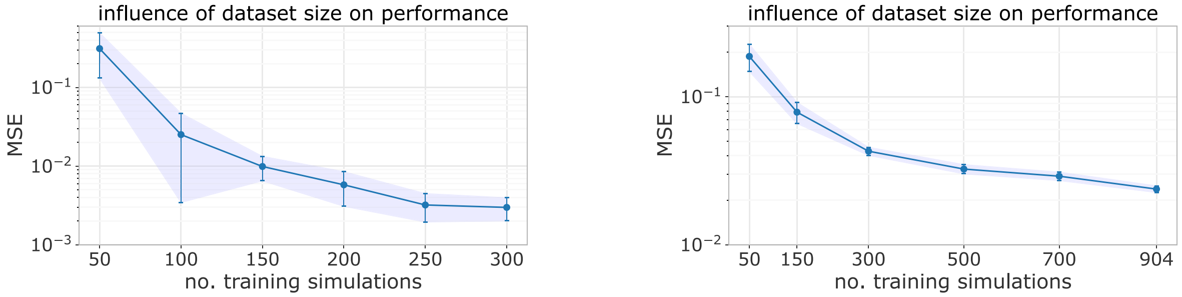


(a) A test-set simulation from the fast transient dataset, here showing only the plasma density and temperature. Corresponding dynamic upstream BCs are depicted in (c), the impurity fraction $\xi_\mathrm{C}$ is set to 0.03 ion/electron (static).

(b) Zoomed-in view of the first transient. DIV1D-NN captures the overall structure well, but there are noticeable artifacts and missing details in the surrogate's solution.

(c) Dynamic boundary conditions for the upstream density $n_\mathrm{u}$ and upstream parallel heat flux $q_{\|\mathrm{u}}$.

**Figure 16.** A test-set simulation from the fast transient dataset: (*a*) evolution of the plasma density and temperature; (*b*) zoomed-in view of a single transient event; and (*c*) the dynamic BCs. The impurity fraction $\xi_\mathrm{C}$ is static at 0.03 ion/electron.



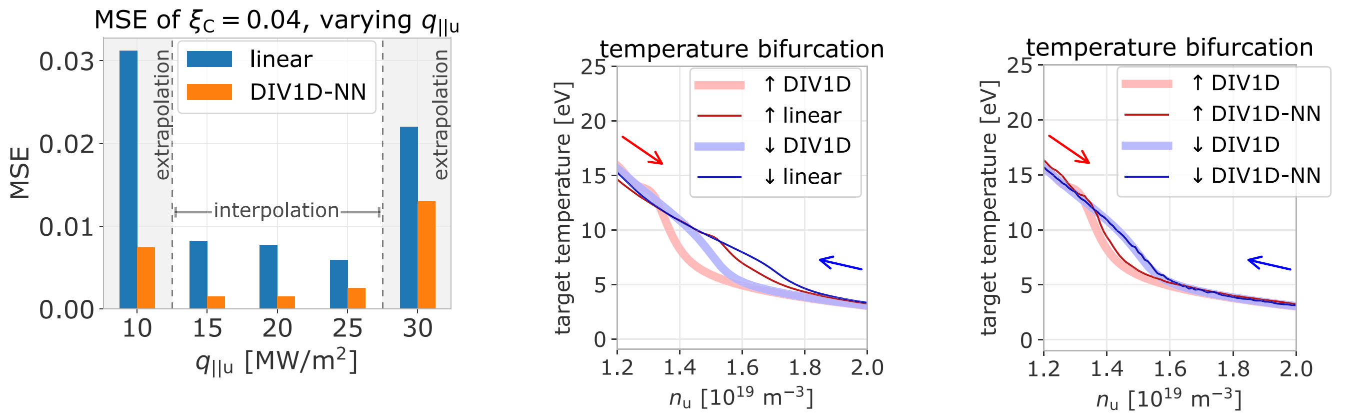
(a) Density ramp dataset

(b) Fast transients dataset

**Figure 17.** MSE of DIV1D-NN as function of dataset size. We use the same validation and test splits as before, but vary the number of train simulations. Each size is evaluated for five different splits; for the full sizes (300 for density ramps and 904 for fast transients) the difference stems from different model initializations. Results are shown as mean ± standard deviation.

dependence on dataset size, we retrain DIV1D-NN on both datasets with varying levels of training samples, using five different subsets for each 'number of samples' setting.

The results for the density ramp data and the fast transient data are plotted in figures 17(*a*) and (*b*), respectively. There is little drop in quality when taking one or two steps down



(a) Evaluation of error when inter- or extrapolating in the parameter space, for both DIV1D-NN and when linearly interpolating surrounding solutions. We use simulations that do not contain either $\xi_C = 0.04$ or $q_{||u} \in \{10, 15, 20, 25, 30\}$ to estimate solutions with these settings.

(b) When linearly interpolating surrounding solutions we cannot capture the bifurcating dynamics correctly.

(c) NN-based interpolation, using proposed surrogate DIV1D-NN, captures the bifurcating dynamics (up to small artifacts).

**Figure 18.** Evaluation of the inter- and extrapolation capability of DIV1D-NN, and a comparison with linear interpolation of surrounding solutions. Errors are given for $\xi_C = 0.04$ and varying $q_{||u}$ in (*a*), where a white background denotes interpolation in the parameter space, and a gray background extrapolation. DIV1D-NN performs best when interpolating in the parameter space and consistently shows benefits over linearly interpolating surrounding solutions. This advantage is illustrated using the bifurcation in the target temperature in (*b*) and (*c*), for 40 ms ramps with $q_{||u} = 20$ MW m$^{-2}$ and $\xi_C = 0.04$.

for both datasets, the surrogates can reach satisfactory performance with relatively few simulations; in the order of 200–250 for the density ramps and about 500 for the fast transients. In general, we hypothesize that the surrogates can accurately match DIV1D with relatively few simulations because we exploit the full spatiotemporal signal of the solutions. For example, for 300 density ramp simulations with 100 gridpoints and an average of ≈1000 timesteps, there are $300 \times 100 \times 1000 \times 4 = 120\,000\,000$ individual points the model uses to find correlations found in DIV1D solutions, a much more substantial sounding dataset.

### 5.6. Evaluation of inter- and extrapolation

The utility of a surrogate lies in its ability to provide *new* solutions fast. Consequently, it is crucial to evaluate the capabilities of the proposed surrogate's inter- and extrapolation capability w.r.t. the parameter space; the extent to which DIV1D-NN can accurately generate solutions in relation to the provided training data.

We evaluate the model performance on the density ramp data by leaving out all solutions with a given upstream parallel heat flux $q_{||u}$ *or* impurity fraction $\xi_C$. We retrain DIV1D-NN with most of the remaining simulations, leaving out a few as validation data. The resulting model is tested on the simulations with the left-out parameters for $q_{||u}$ and $\xi_C$.

Additionally, we consider linear interpolation between existing solutions as a baseline. New solutions are formed by interpolating between solutions for the surrounding values of $q_{||u}$ and $\xi_C$ with identical ramp speed $\dot{n}_u$, or extrapolating from the two closest values if we cannot interpolate. In practice we are lenient towards this method: Not all of these simulations are necessarily available and used as DIV1D-NN's training data, but we assume they are always available for the linear interpolation baseline.

Results for $\xi_C = 0.04$ ion/electron and all values of $q_{||u}$ are provided in figure 18(*a*). As expected, the results are much better when interpolating within the parameter space, compared to extrapolating outside of this range. In general, these surrogate modeling techniques are ill-suited for extrapolating (far) beyond the training data. DIV1D-NN outperforms linear interpolation in all cases by a wide margin, although we note that in some parameter extrapolation settings (not depicted here) linear extrapolation outperforms DIV1D-NN; for details we refer to appendix E. Evaluating non-linear dynamics, figures 18(*b*) and (*c*) depict the bifurcation in the target temperature [18] for linear interpolation and for DIV1D-NN, respectively. The complete mismatch for linear interpolation shows a clear benefit of using neural PDE surrogates.

## 6. Conclusions and discussion

We have presented the application and extension of neural PDE surrogate techniques for building a fast dynamic 1D surrogate model of divertor plasmas. We demonstrated the application of an autoregressive NN-based model, which approximates solutions by learning a time-stepping operator that evolves the state of the system, following the structure proposed in [12]. Within this framework, we investigated and extended five state-of-the-art NN architectures to approximate said operator. The evaluation showed that many methods can accurately approximate solutions, most notably the Dilated Residual Network [16]. We proposed surrogate architecture

DIV1D-NN, which showed the best trade-off between error and computation time. DIV1D-NN can simulate density ramp dynamics faster than real-time, simulating a time evolution of 2 ms in ≈0.63 ms of wall-clock time.

Due to explicitly taking into account the dynamics of the plasma, we are able to recover non-linear time-dependent phenomena, demonstrated through the bifurcation in the target temperature. Further investigation also showed the recovery of properties and structures such as the roll-over in the target ion flux and the location of the emission front. An introductory evaluation of fast transient dynamics shows that the surrogate can reproduce higher-frequency dynamics, although there are improvements to be made in this area.

The aforementioned surrogate models could be trained with relatively few simulations, in the order of hundreds for density ramps and about 500 for fast transients. Additionally, we have evaluated the inter- and extrapolation capability of DIV1D-NN, showing its suitability for non-linearly interpolating within the parameter space of the training set.

One limitation of this work is that the dynamics in the divertor (along the magnetic field line) generally are very fast. In the density ramp dataset, the slowest ramp rates result in quasi-stationary profiles as a function of upstream conditions. With very fast perturbations, as found in the fast transient dataset, the simulated divertor plasma returned to steady-state conditions in the order of milliseconds. As such, many time dependencies are captured in only a few solution blocks (of 2 ms each), the ability to model long-range time dependencies cannot be inferred from the conducted evaluations.

### *6.1. Future work*

The presented DIV1D-NN surrogate represents a real-time model of realistic TCV divertor plasmas. Of interest is the application in real-world use cases. For example, one could explore coupling a fast neural PDE surrogate in a flight simulator-setting [57–59], replacing lower-fidelity approximations. Another promising setting is real-time control, for example, to exploit real-time high-fidelity estimates of the plasma evolution in exhaust control schemes [17] or in advanced control algorithms that require a model to be evaluated in real-time [60]. To improve a surrogate's utility in this setting one could explore constraining its dynamics, for example by learning a coordinate transform in which dynamics are linear [61, 62]. Furthermore, one can envision combining diagnostic measurements with fast state estimates in a Kalman filter-like setting [63]; especially with potentially limited diagnostics in future reactors [64] fast high-fidelity plasma state estimates could become vital.

To improve the approximation quality of the surrogate model, one can explore a diverse set of directions. Different methods of conditioning the NN could be explored, for example, attention-based methods have shown strong performance in different domains [65]. Another angle is to use alternative approaches to deal with the distribution shift problem (section 4.2), one being to inject noise into the training inputs [16]. To better exploit dynamics found in data, one can structure networks better suited for geometrical transformations found in dynamical PDE solutions with stronger geometric priors [66]. One can consider exploiting additional information following from physics laws [67, 68], as opposed to the primarily data-driven approach presented here. Stepping away from full surrogate models, one can explore hybrid techniques that combine machine learning and classical numerical methods [69] to accelerate existing physics-based codes. Finally, the proposed techniques make deterministic single-point predictions not taking into account any uncertainties. In settings with turbulent dynamics or with experimental measurements it is crucial to explicitly account for uncertainty, for example, due to unresolved turbulence scales or measurement uncertainty [70].

In a broader context, one can apply the discussed techniques to different domains and modalities. Numerical simulation of tokamak plasmas has proven vital to the development and operation of tokamak devices in various plasma regions and on many levels of fidelity [6, 7, 9, 71–75], and has shown reasonable agreement with experimental observations in many cases [76–79]. However, in a significant number of codes, the computational cost is prohibitively expensive. If a dataset with a sufficient number of simulations could be generated, the presented surrogate modeling techniques could be applied in order to generate many new simulations at a fraction of the cost or to simulate in time-sensitive control contexts. Since the presented techniques have shown good results using only hundreds of simulations, one could potentially apply them to much more expensive codes for which this number of simulations is obtainable (see e.g. [80]). In a similar vein, one could further explore the presented methods for creating data-driven simulators directly based on real-world physics, using the vast amount of experimental data that has been collected for a wide array of tokamaks.

## Acknowledgments

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200—EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them. This work made use of the Dutch national e-infrastructure with the support of the SURF Cooperative using Grant No. EINF-3557.

## Appendix A. Architecture figures

This appendix contains more detailed illustrations of the architectures described in section 4.3. Figures A.1–A.5 contain illustrations for DRN [16], UNet [14], FNO [15], MP-PDE [12] and FT [13], respectively. All but the UNet illustrate a single hidden block that is repeated, the UNet illustration depicts the entire architecture.

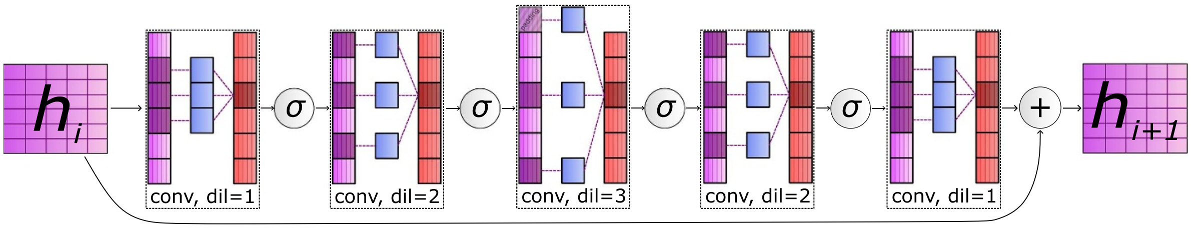


**Figure A.1.** Dilated Residual Network [16]: The **DRN** applies a sequence of dilated convolutions with varying dilation rates.

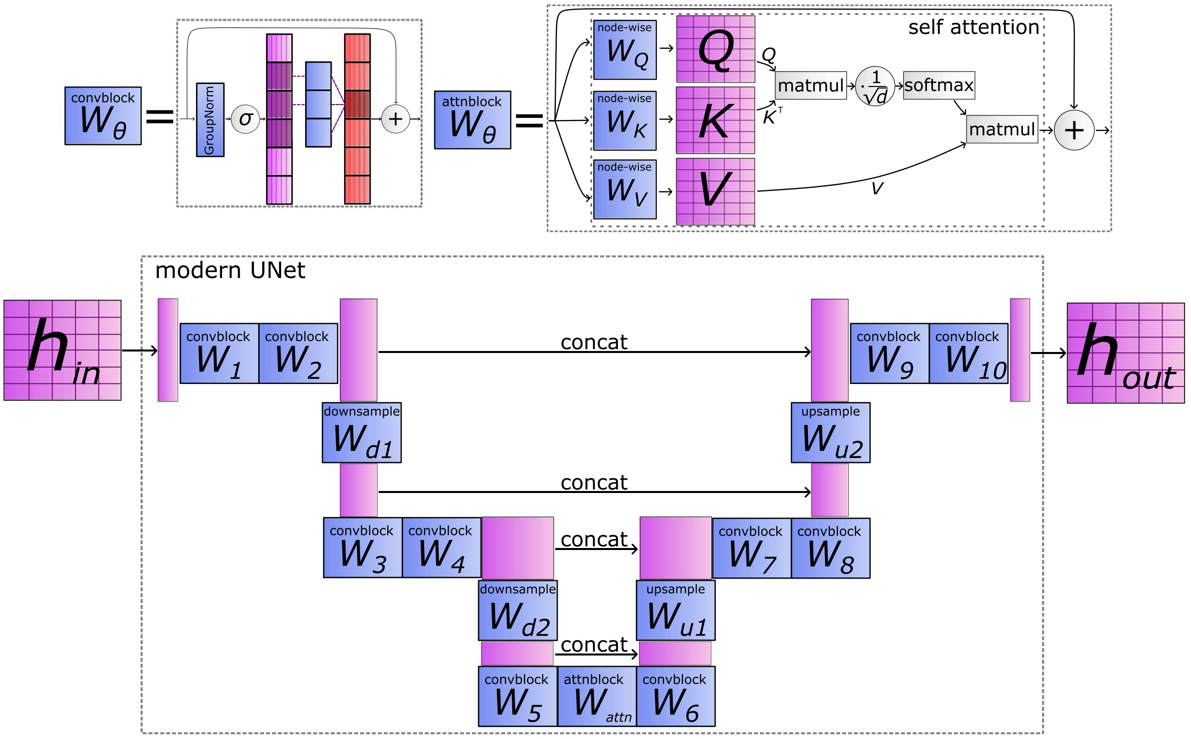


**Figure A.2.** Modern UNet [14]: The **UNet** applies convolutions on multiple spatial scales, with connections on the same scales.

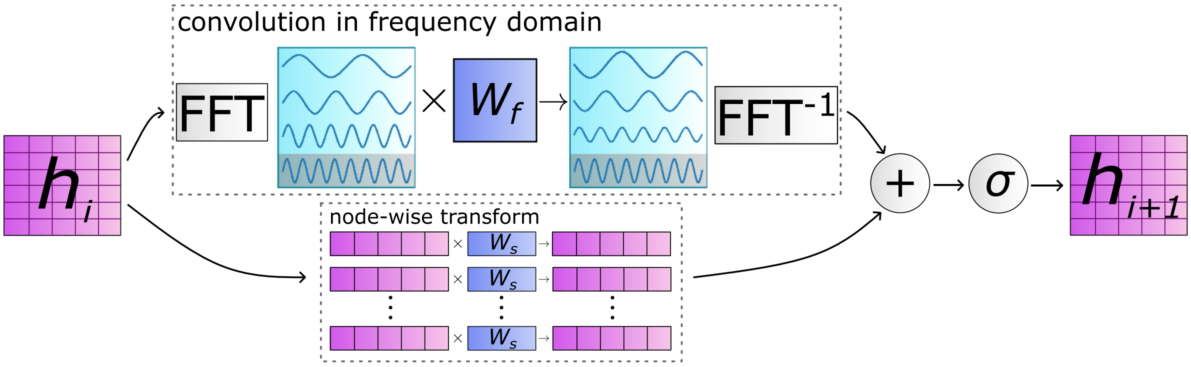
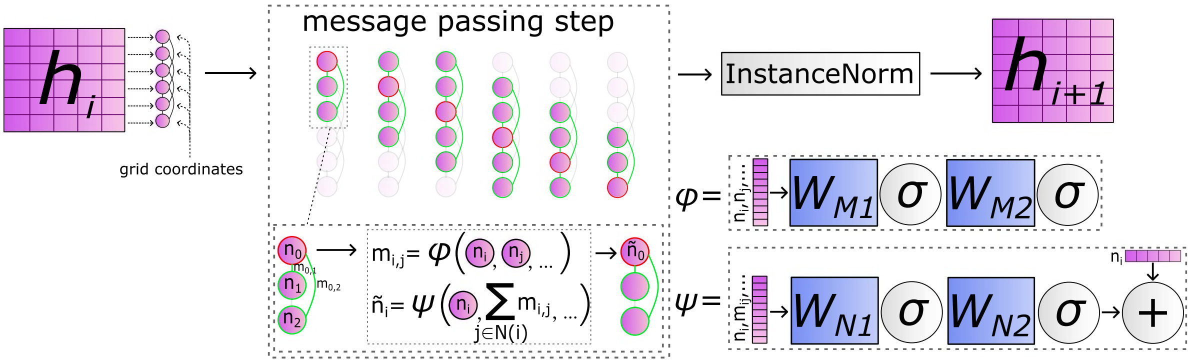


**Figure A.3.** Fourier Neural Operator [15]: The **FNO** parameterizes a convolution operator in Fourier space.



$h_i$
grid coordinates
message passing step
InstanceNorm
$h_{i+1}$
$\varphi=$
$W_{M1}$ $\sigma$ $W_{M2}$ $\sigma$
$\psi=$
$W_{N1}$ $\sigma$ $W_{N2}$ $\sigma$ $+$
$n_0$ $n_1$ $n_2$ $m_{0,1}$ $m_{0,2}$
$m_{i,j}=\varphi(n_i, n_j, ...)$
$\tilde{n}_i=\psi(n_i, \sum_{j\in N(i)} m_{i,j}, ...)$
$\tilde{n}_0$
$n_i$

**Figure A.4.** Message Passing PDE Solver [12]: The **MP-PDE** solver updates a point's representation using information from itself and its neighbors.

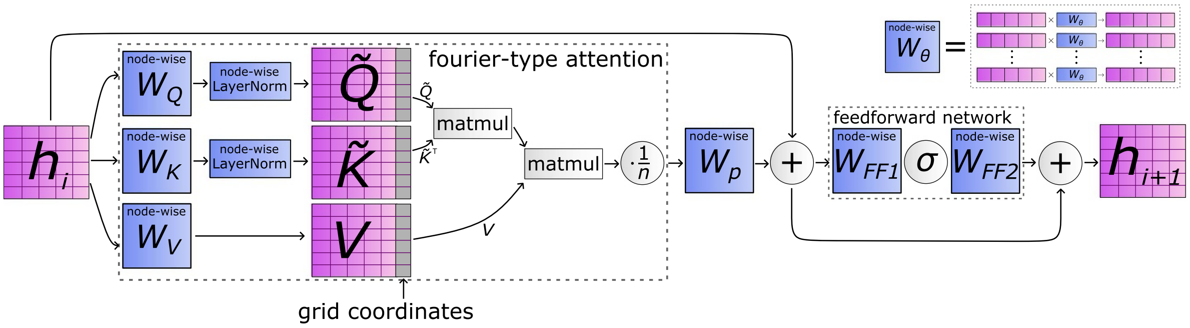


**Figure A.5.** Transformer with Fourier-type Attention [13]: The **FT** updates a point's representation with the product of values $\mathbf{V}$ and attention score $\tilde{\mathbf{Q}}\tilde{\mathbf{K}}^\top$.

## Appendix B. Time-Adjusted Batch Sampling

In this appendix we elaborate on the time-adjusted batch sampling strategy as introduced in section 4.4 and provide evaluations of its utility. The best settings are used to train the set of models found in section 5.

For training, we sample batches of datapoints by first sampling the unrolling time $t$ from $p(t)$, and subsequently sample batch items $x$ with adjusted probabilities given this unrolling time from $p(x|t)$. The goal is to both adhere to the unrolling time distribution and to sample simulations uniformly, see also equation (11).

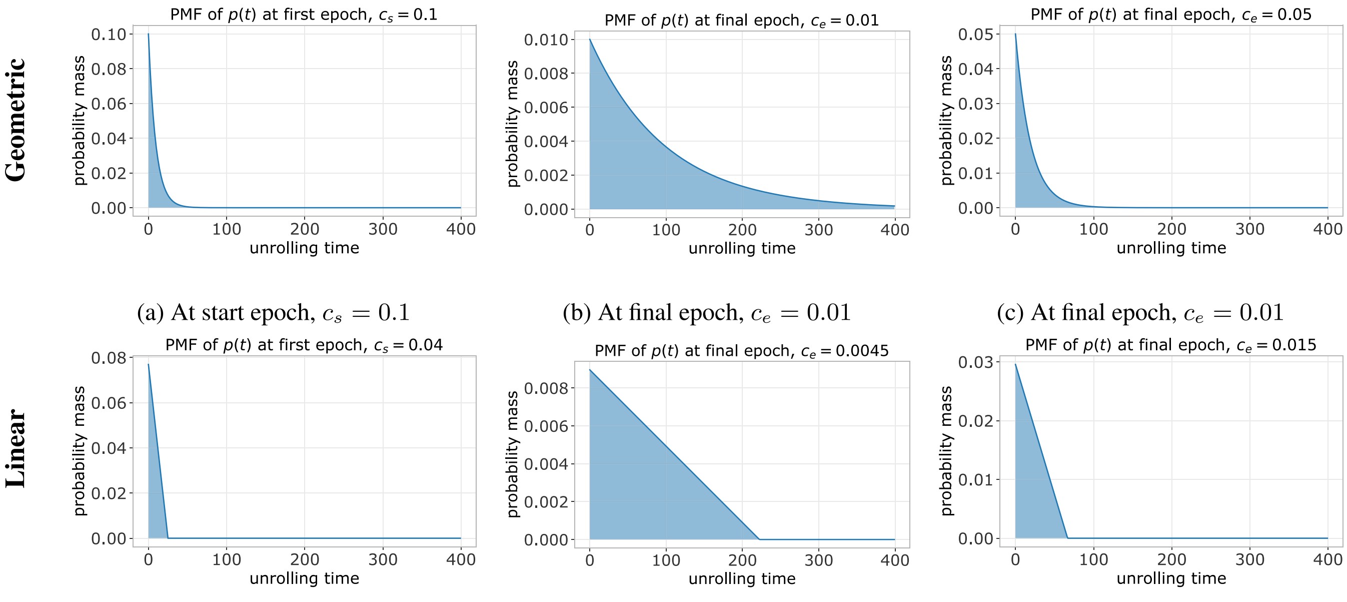
In short, we compute $p(x|t)$ using intervals of simulation time windows $Nt$, by placing simulations into time-based bins and adjusting sample probabilities based on the probability of sampling in an interval given $p(t)$. We start at the bin with the shortest simulations, and simply assign these shortest simulations the adjusted probability such that the marginalized probability is $\frac{1}{\mathrm{N}}$. These items are no longer considered as they cannot be used when unrolling more steps in 'longer' bins. The process is repeated with the now-shortest bin and its simulations while adjusting for the previously set probabilities, which is again repeated until all items are parsed.

The method for computing $p(x|t)$ is described in pseudocode 1 in more detail. Distribution $p(x|t)$ is computed using a list of simulation times `times`, the block length `w` and the cumulative distribution function of $p(t)$. Items are first placed in bins (line 2). These bins are computed by the number of unrollings one can do, i.e. by rounding the $Nt$ of each simulation down to a multiple of block length $w$. Each bin is defined by the items falling in this bin (`bins`$_i$`.items`) and the number of timesteps $t$ it can be unrolled (`bins`$_i$`.time`, a multiple of $w$), and is sorted on ascending time. Lines 6–15 describe the main procedure of assigning probabilities to items in bins. We start at the smallest bin `bins`$_1$, as items in this bin can only be used when unrolling for at most `bins`$_1$`.time`. Item probabilities are computed in line 8: Given the unassigned probability mass that is available to this item (`bin_prob`—from this bin, and `res_prob`—from smaller bins, which is 0 for the first iteration), the expectation of sampling it will be $\frac{1}{N}$. This procedure is repeated for all bins up until the last one. For the last bin we simply assign the remaining probability mass (line 16). Finally, in lines 17–21, the output object is post-processed to distribute the residual probability mass up to chain. We return the bins and probability assignments within the bin, defining $p(x|t)$.

To evaluate this approach we try different unrolling distributions $p(t)$ and do a comparison with other padding-free baselines (that also do not waste the majority of computation time, see section 4.4). As probability mass function (PMF) of $p(t)$ we try the geometric distribution and a simple linear function. At the start of training, distribution parameters are chosen to make the PMFs steep, i.e. putting almost all weight towards short unrolling times. They are then made more shallow over time: As training goes on, we do longer and longer unrollings. We use the same schedule for adjusting the distribution

**Pseudocode 1:** Compute $p(x|t)$ Algorithm.

```
Function compute_probabilities ($times, w, CDF$):
    bins $\leftarrow sort(create\_bins(\texttt{times}, \texttt{w}))$    // Binned by time $Nt$, with variables $\texttt{bins}_i\texttt{.time}$ and $\texttt{bins}_i\texttt{.items}$
    $N \leftarrow |\texttt{bins}|$
    expected $\leftarrow \frac{1}{|\texttt{times}|}$    // All simulations should be sampled uniformly
    res_prob $\leftarrow 0$    // Probability mass that is unassigned in already-parsed bins
    for $i \leftarrow 1$ to $N-1$ do
        bin_prob $\leftarrow CDF(\texttt{bins}_i\texttt{.time}) - CDF(\texttt{bins}_{i-1}\texttt{.time})$    // Probability $t$ falls in this bin
        item_prob $\leftarrow \frac{\texttt{expected}}{\texttt{bin\_prob}+\texttt{res\_prob}}$    // Probability needed from this bin + previous residuals
        bin_items $\leftarrow \{j : \texttt{item\_prob} \mid j \in \texttt{bins}_i\texttt{.items}\}$    // Set probs for bin items
        other_prob $\leftarrow 1 - \sum \texttt{bin\_items}$    // Get bin residual
        if other_prob $< 0$ then    // Verify a valid assignment is possible
            error: Bin residual $< 0$, no valid assignment possible
        bin_items ['other'] $\leftarrow$ other_prob    // Store bin residual
        $\texttt{bins}_i$ .probs $\leftarrow$ bin_items    // Save bin probabilities
        // Update res_prob: Add residual from $\texttt{bins}_i$, remove residual consumed by $\texttt{bins}_i\texttt{.items}$
        res_prob $\leftarrow$ res_prob + bin_prob $\cdot$ other_prob $-$ res_prob $\cdot (1 - \texttt{other\_prob})$
    $\texttt{bins}_N$.probs $\leftarrow \{j : \frac{1}{|\texttt{bins}_N\texttt{.items}|} \mid j \in \texttt{bins}_N\texttt{.items}\})$    // Probabilities for final bin (no residual)
    for $i \leftarrow N-1$ to $1$ do    // Distribute residuals up the chain
        $idx \leftarrow \cup_{j=i+1}^{N} \texttt{bins}_j\texttt{.items}$    // Indices of items in longer bins
        foreach $j \in idx$ do
            $\texttt{bins}_i\texttt{.probs}[j] \leftarrow \texttt{bins}_i\texttt{.probs}[j] + \texttt{bins}_{i+1}\texttt{.probs}[j] \cdot \texttt{bins}_i\texttt{.probs}$ ['other']
        delete $\texttt{bins}_i$.probs ['other']
    return bins    // Variables $\texttt{bins}_i\texttt{.time}$, $\texttt{bins}_i\texttt{.items}$ and $\texttt{bins}_i\texttt{.probs}$ (probablity mass assignment of $p(x|t)$)
```



(a) At start epoch, $c_s = 0.1$ (b) At final epoch, $c_e = 0.01$ (c) At final epoch, $c_e = 0.01$

(d) At start epoch, $c_s = 0.04$ (e) At final epoch, $c_e = 0.0045$ (f) At final epoch, $c_e = 0.0015$

**Figure B.1.** PMFs we evaluate for sampling unrolling time $t \sim p(t)$: The geometric distribution and a linear function. Coefficient $c$ is decayed from $c_s$ to $c_e$, we try two end coefficients per distribution.

coefficient, denoted as $c$, and try two coefficients per distribution family. A depiction of these distributions is provided in figure B.1, with the coefficient schedule in figure B.2.

As baseline we consider two methods for sampling batches: (1) Uniformly sampling simulations in the minibatch and then sampling an unrolling time that fits given the batch items, and (2) sampling an unrolling time and then uniformly sampling simulations that fit this length. The downsides are that for (1) we have no guarantees on the sampling time distribution and for (2) we have no guarantees on sampling simulations uniformly. For both methods we uniformly sample unrolling times, either up until 160 timesteps or up until 400. This sampling window starts at 0 and is increased as

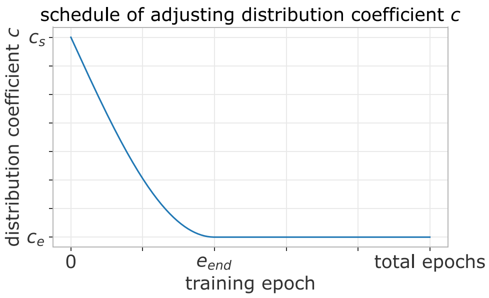


**Figure B.2.** Schedule of coefficient $c$ during training. $c$ is decayed from $c_s$ to $c_e$ through the bottom half of a sine wave until epoch $e_{\text{end}}$ (here at 40% of training), after which it is kept fixed at $c_e$.

**Table B1.** MSE on standardized data, full simulation rollouts on the validation and test set. Cells are colored by the error, with green being better. The best scores are marked in bold.

| Sampling strategy | Validation | Test |
|---|---|---|
| AdjSampling-Geometric$_{0.01}$ | 0.001 37 | 0.002 91 |
| AdjSampling-Geometric$_{0.01}$-NR | 0.001 21 | 0.003 36 |
| AdjSampling-Geometric$_{0.05}$ | 0.009 41 | 0.010 26 |
| AdjSampling-Geometric$_{0.05}$-NR | 0.001 74 | 0.003 36 |
| AdjSampling-Linear$_{0.0045}$ | **0.000 84** | **0.002 54** |
| AdjSampling-Linear$_{0.0045}$-NR | 0.002 80 | 0.004 43 |
| AdjSampling-Linear$_{0.015}$ | 0.007 98 | 0.021 69 |
| AdjSampling-Linear$_{0.015}$-NR | 0.019 38 | 0.008 29 |
| SimFirst-Unrolling$_{160}$ | 0.008 99 | 0.015 36 |
| SimFirst-Unrolling$_{160}$-NR | 0.072 73 | 0.556 32 |
| SimFirst-Unrolling$_{400}$ | 0.005 04 | 0.013 51 |
| SimFirst-Unrolling$_{400}$-NR | 0.007 86 | 0.008 74 |
| TimeFirst-Unrolling$_{160}$ | 0.006 27 | 0.017 61 |
| TimeFirst-Unrolling$_{160}$-NR | 0.090 00 | 0.019 75 |
| TimeFirst-Unrolling$_{400}$ | 0.004 72 | 0.028 93 |
| TimeFirst-Unrolling$_{400}$-NR | 0.015 72 | 0.007 36 |

training goes on, at 20 timesteps for each 5% of training epochs (160 timesteps being reached at 40%, 400 at the end of training).

All methods are tested by sampling batches in an epoch either with replacement or with no replacement. The latter setting, denoted with **–NR**, ensures we sample each item once per epoch, enforcing the desired expectation of $\frac{1}{N}$. However, this setting can bias the eligible unrolling windows to shorter windows.

The methods are evaluated before the final experiments (section 5) in order to pick a setting that we choose for all models. We investigate using the density ramp dataset and use reference settings that we found to work reasonably well: Train for 10 000 epochs with batch size 16 (where an epoch is one pass over each simulation, for a single random start point and its unrolling window). The model consists of an element-wise encoder, a Dilated Residual Network as processor (8 blocks; 512 hidden features; kernel size 5), and a time-wise convolution as decoder. We compute solutions in blocks of $w = 20$ timesteps (2 ms real time).

The results are reported in table B1. We choose the settings with the best results on the validation set for further experiments, the test set results are reported only for completeness. The adjusted sampling methods generally result in better models, with a 'wide' linear distribution for the unrolling distribution $p(t)$ working best (AdjSampling-Linear$_{0.0045}$, figure B.1($e$)).

## Appendix C. Tables: density ramp results

In this appendix we provide full results for the evaluated neural PDE surrogate architectures for the density ramp data. All architectures follow the encode-process-decode structure, where *encoder*: $\mathbf{ub}^{i-w:i} \in \mathbb{R}^{Nx\times w\times 4} \to h_{\text{in}} \in \mathbb{R}^{Nx\times D}$ maps the input block to a hidden representation, *processor*: $h_{\text{in}} \in \mathbb{R}^{Nx\times D} \to h_{\text{out}} \in \mathbb{R}^{Nx\times D}$ transforms this hidden representation, and *decoder*: $h_{\text{out}} \in \mathbb{R}^{Nx\times D} \to \widetilde{\mathbf{ub}}^{i:i+w} \in \mathbb{R}^{Nx\times w\times 4}$ maps this representation back to the next block, all conditioned by $(\mathbf{b}_{\text{s}}, \mathbf{b}_{\text{d}}^{\mathbf{t}_{i:i+w}}, \mathbf{c})$; see also sections 4.3 and 4.4.

Architectures are found by iteratively reducing the size of an architecture of 'maximum width' and 'maximum depth', where the former emphasises using many hidden dimensions, and the latter emphasises using many layers. For each architecture a set of hyperparameters is iteratively updated (up to some minimum values) until an architecture is found with an inference time below $\{0.5, 1, 2, 4, 6, 8, 10\}$ ms[10], computed as a forward pass of batch size 1 on an NVIDIA A100 40GB GPU with an Intel Xeon Platinum 8360Y CPU. The tested hyperparameters and other settings are as follows.

For all architectures the number of hidden dimensions is varied, which is denoted with $D$. All architectures besides the UNet consist of repeating layers (with identical architectures but non-shared parameters), where the number of layers is denoted with $L$. For all models, three combinations of an encoder/decoder/dynamic BC encoder are considered, denoted with **EncDec**. These are as follows:

- **EncDec$_{\mathbf{B1}}$**. *Encoder*: Point-wise non-linear map to $D$ dimensions as in [12], with Swish activations [81] and 1 hidden layer. *Decoder*: Feature-wise non-linear convolution mapping back to a time-block of solutions as in [12], with Swish activations and 1 hidden layer. *BC encoder*: Non-linear convolution over the timesteps in the block, flattened and mapped to an embedding of 8 dimensions, using 1 hidden layer with GELU activations [82].
- **EncDec$_{\mathbf{B2}}$**. *Encoder*: Point-wise non-linear map to $D$ dimensions with GELU activations and 1 hidden layer. *Decoder*: Point-wise mapping from $D \to 12 \times w$ features, followed by a feature-wise non-linear convolution as in EncDec$_{\text{B1}}$, but with GELU activations instead. *BC encoder*: Same as EncDec$_{\text{B1}}$.
- **EncDec$_{\mathbf{S}}$**. *Encoder*: Grid-wise linear convolution with kernel size 3. *Decoder*: Grid-wise linear convolution with kernel size 3. *BC encoder*: Non-linear convolution over the

[10] Note that not all inference times are reached for all models, due to the smallest architecture still exceeding the smallest desired inference times in most cases.

**Table C1.** Complete results on test data for the density ramp dataset. Architectures are found by optimizing hyperparameters to a set of inference times between 0.5 to 10 ms. Evaluations are provided for three error metrics, where green indicates a lower error, alongside the inference speed for two types of hardware, where red indicates a larger inference cost. A dash indicates that the model converged to noise, filtered as the error being greater than 5.

| | | Error metrics | | | Compute time (ms) (per 2 ms) | |
|---|---|---|---|---|---|---|
| | Model$_{\mathrm{configuration}}$ | MSE | MSE$_{t<500}$ | MSE$_{\mathrm{single}}$ | GPU | CPU$_{\mathrm{1\text{-}core}}$ |
| depth | DRN$_{\mathrm{L:8,D:832,EncDec_{B1}}}$ | 0.001 10 | 0.000 78 | 0.000 31 | 9.237 71 | 521.604 |
| | DRN$_{\mathrm{L:6,D:832,EncDec_{B2}}}$ | 0.001 07 | 0.000 65 | 0.000 23 | 7.189 71 | 395.983 |
| | DRN$_{\mathrm{L:6,D:832,EncDec_{B1}}}$ | 0.002 99 | 0.000 56 | 0.000 27 | 7.137 24 | 394.271 |
| | DRN$_{\mathrm{L:7,D:768,EncDec_{B1}}}$ | 0.002 55 | 0.000 92 | 0.000 32 | 5.166 61 | 398.770 |
| | DRN$_{\mathrm{L:4,D:704,EncDec_{B2}}}$ | 0.000 86 | 0.000 46 | 0.000 19 | 3.552 01 | 190.693 |
| | DRN$_{\mathrm{L:4,D:704,EncDec_{B1}}}$ | 0.001 33 | 0.000 61 | 0.000 69 | 3.420 41 | 188.738 |
| | DRN$_{\mathrm{L:2,D:128,EncDec_{S}}}$ | 2.287 18 | 0.012 85 | 0.000 76 | 1.785 69 | 10.2031 |
| | DRN$_{\mathrm{L:1,D:64,EncDec_{B2}}}$ | 0.093 58 | 0.042 52 | 0.007 54 | 1.432 29 | 3.391 29 |
| | DRN$_{\mathrm{L:1,D:64,EncDec_{B1}}}$ | — | 0.064 12 | 0.004 19 | 1.409 84 | 3.224 18 |
| | DRN$_{\mathrm{L:1,D:64,EncDec_{B1}}}$ | 0.221 27 | 0.057 35 | 0.003 78 | 1.405 91 | 3.226 75 |
| | DRN$_{\mathrm{L:1,D:128,EncDec_{S}}}$ | — | 0.031 24 | 0.001 04 | 1.188 95 | 5.555 77 |
| | DRN$_{\mathrm{L:1,D:64,EncDec_{S}}}$ | — | 0.076 65 | 0.003 20 | 1.108 79 | 2.266 16 |
| width | DRN$_{\mathrm{L:6,D:1408,EncDec_{B1}}}$ | 0.004 64 | 0.000 70 | 0.000 32 | 9.596 35 | 1214.41 |
| | DRN$_{\mathrm{L:4,D:1536,EncDec_{B1}}}$ | 0.001 39 | 0.000 56 | 0.000 23 | 7.524 27 | 941.731 |
| | DRN$_{\mathrm{L:3,D:1536,EncDec_{B1}}}$ | 0.001 18 | 0.000 93 | 0.000 56 | 5.842 62 | 700.517 |
| | DRN$_{\mathrm{L:3,D:1152,EncDec_{B2}}}$ | 0.001 12 | 0.000 58 | 0.000 27 | 3.947 19 | 394.983 |
| | DRN$_{\mathrm{L:3,D:1152,EncDec_{B1}}}$ | 0.000 79 | 0.000 37 | 0.000 20 | 3.898 98 | 399.986 |
| | DRN$_{\mathrm{L:1,D:1152,EncDec_{B2}}}$ | 0.002 37 | 0.000 43 | 0.000 21 | 1.853 89 | 143.139 |
| | DRN$_{\mathrm{L:1,D:1152,EncDec_{B1}}}$ | 0.001 92 | 0.000 41 | 0.000 22 | 1.807 36 | 140.160 |
| | DRN$_{\mathrm{L:2,D:128,EncDec_{S}}}$ | — | 0.021 25 | 0.000 83 | 1.791 84 | 10.2170 |
| | DRN$_{\mathrm{L:1,D:128,EncDec_{B2}}}$ | 0.031 62 | 0.008 35 | 0.001 69 | 1.660 56 | 8.408 28 |
| | DRN$_{\mathrm{L:1,D:128,EncDec_{B1}}}$ | 0.048 81 | 0.018 62 | 0.001 95 | 1.570 86 | 8.099 00 |
| | DRN$_{\mathrm{L:1,D:128,EncDec_{S}}}$ | — | 0.058 37 | 0.001 13 | 1.195 50 | 5.572 75 |
| | DRN$_{\mathrm{L:1,D:128,EncDec_{S}}}$ | 0.382 69 | 0.033 10 | 0.001 11 | 1.193 53 | 5.566 22 |
| depth | UNet$_{\mathrm{D:40,Depth:3,EncDec_{B2}}}$ | 0.011 26 | 0.008 41 | 0.015 52 | 9.267 28 | 7.446 07 |
| | UNet$_{\mathrm{D:40,Depth:3,EncDec_{B1}}}$ | 0.231 86 | 0.132 11 | 0.015 21 | 9.139 28 | 7.318 69 |
| | UNet$_{\mathrm{D:48,Depth:2,EncDec_{B2}}}$ | 0.015 62 | 0.006 94 | 0.005 89 | 7.011 86 | 6.609 78 |
| | UNet$_{\mathrm{D:48,Depth:2,EncDec_{B1}}}$ | 0.113 21 | 0.034 37 | 0.015 92 | 6.947 39 | 6.498 98 |
| | UNet$_{\mathrm{D:80,Depth:1,EncDec_{B2}}}$ | 0.002 83 | 0.003 16 | 0.005 47 | 4.651 21 | 7.688 50 |
| | UNet$_{\mathrm{D:80,Depth:1,EncDec_{B1}}}$ | 0.008 87 | 0.006 86 | 0.004 39 | 4.623 07 | 7.498 70 |
| | UNet$_{\mathrm{D:32,Depth:1,EncDec_{B1}}}$ | 0.498 90 | 0.220 82 | 0.015 46 | 4.082 93 | 3.632 59 |
| | UNet$_{\mathrm{D:32,Depth:1,EncDec_{S}}}$ | 0.041 01 | 0.013 89 | 0.008 62 | 3.843 32 | 3.268 67 |
| width | UNet$_{\mathrm{D:40,Depth:3,EncDec_{B2}}}$ | 0.008 73 | 0.007 91 | 0.017 43 | 9.210 76 | 7.428 98 |
| | UNet$_{\mathrm{D:40,Depth:3,EncDec_{B1}}}$ | 0.278 84 | 0.107 58 | 0.015 52 | 9.121 14 | 7.320 25 |
| | UNet$_{\mathrm{D:48,Depth:2,EncDec_{B2}}}$ | 0.011 97 | 0.005 92 | 0.007 15 | 6.979 71 | 6.597 95 |
| | UNet$_{\mathrm{D:48,Depth:2,EncDec_{B1}}}$ | 0.098 92 | 0.035 11 | 0.016 39 | 6.922 77 | 6.505 37 |
| | UNet$_{\mathrm{D:128,Depth:1,EncDec_{B2}}}$ | 0.001 36 | 0.001 67 | 0.001 27 | 4.847 94 | 12.5765 |
| | UNet$_{\mathrm{D:128,Depth:1,EncDec_{B1}}}$ | 0.001 14 | 0.001 37 | 0.00819 | 4.745 99 | 12.2107 |
| | UNet$_{\mathrm{D:32,Depth:1,EncDec_{B1}}}$ | 0.437 70 | 0.191 21 | 0.015 19 | 4.075 27 | 3.627 07 |
| | UNet$_{\mathrm{D:32,Depth:1,EncDec_{S}}}$ | 0.020 02 | 0.012 30 | 0.00846 | 3.841 60 | 3.266 04 |
| depth | FNO$_{\mathrm{L:12,D:512,M:44,EncDec_{B1}}}$ | 0.007 07 | 0.001 04 | 0.000 39 | 9.371 48 | 729.302 |
| | FNO$_{\mathrm{L:10,D:512,M:42,EncDec_{B2}}}$ | 0.002 53 | 0.000 86 | 0.000 49 | 7.674 63 | 590.495 |
| | FNO$_{\mathrm{L:10,D:512,M:42,EncDec_{B1}}}$ | 0.004 75 | 0.000 88 | 0.000 27 | 7.634 94 | 591.633 |
| | FNO$_{\mathrm{L:10,D:480,M:42,EncDec_{B2}}}$ | 0.008 16 | 0.000 93 | 0.000 27 | 5.710 19 | 496.134 |
| | FNO$_{\mathrm{L:10,D:480,M:42,EncDec_{B1}}}$ | 0.005 24 | 0.001 09 | 0.000 38 | 5.669 40 | 493.843 |
| | FNO$_{\mathrm{L:9,D:416,M:38,EncDec_{B2}}}$ | 0.009 92 | 0.001 32 | 0.000 28 | 3.813 25 | 247.019 |
| | FNO$_{\mathrm{L:9,D:416,M:38,EncDec_{B1}}}$ | 0.059 83 | 0.001 30 | 0.000 28 | 3.713 39 | 246.963 |
| | FNO$_{\mathrm{L:3,D:160,M:16,EncDec_{B1}}}$ | — | 0.031 78 | 0.003 08 | 1.856 02 | 5.612 18 |
| | FNO$_{\mathrm{L:4,D:64,M:8,EncDec_{S}}}$ | — | 0.162 40 | 0.002 54 | 1.723 11 | 1.908 21 |
| | FNO$_{\mathrm{L:2,D:96,M:10,EncDec_{S}}}$ | 3.220 16 | 0.244 84 | 0.003 73 | 1.217 33 | 1.841 18 |
| | FNO$_{\mathrm{L:1,D:32,M:8,EncDec_{B1}}}$ | 2.114 00 | 1.342 09 | 0.017 61 | 1.045 09 | 1.055 74 |
| | FNO$_{\mathrm{L:1,D:64,M:8,EncDec_{S}}}$ | 1.436 78 | 1.359 91 | 0.015 47 | 0.853 36 | 0.941 93 |
| | FNO$_{\mathrm{L:1,D:32,M:8,EncDec_{S}}}$ | — | 0.416 43 | 0.018 55 | 0.829 32 | 0.726 37 |

(Continued.)

**Table C1.** (Continued.)

| | | Error metrics | | | Compute time (ms) (per 2 ms) | |
|---|---|---|---|---|---|---|
| | Model$_{\text{configuration}}$ | MSE | MSE$_{t<500}$ | MSE$_{\text{single}}$ | GPU | CPU$_{\text{1-core}}$ |
| width | FNO$_{\text{L:5,D:768,M:42,EncDec}_{\text{B1}}}$ | 1.790 74 | 0.001 29 | 0.000 59 | 9.739 18 | 703.420 |
| | FNO$_{\text{L:5,D:704,M:40,EncDec}_{\text{B2}}}$ | 0.485 78 | 0.001 44 | 0.000 29 | 7.899 42 | 564.436 |
| | FNO$_{\text{L:5,D:704,M:40,EncDec}_{\text{B1}}}$ | 0.074 51 | 0.001 49 | 0.000 31 | 7.854 74 | 561.600 |
| | FNO$_{\text{L:5,D:640,M:34,EncDec}_{\text{B1}}}$ | 3.020 45 | 0.005 82 | 0.002 73 | 5.741 61 | 397.831 |
| | FNO$_{\text{L:4,D:576,M:30,EncDec}_{\text{B2}}}$ | 0.309 59 | 0.002 82 | 0.000 37 | 3.631 96 | 229.023 |
| | FNO$_{\text{L:4,D:576,M:30,EncDec}_{\text{B1}}}$ | — | 0.002 75 | 0.000 62 | 3.595 67 | 227.535 |
| | FNO$_{\text{L:3,D:384,M:20,EncDec}_{\text{B1}}}$ | 1.192 78 | 0.014 21 | 0.001 07 | 1.864 34 | 54.7761 |
| | FNO$_{\text{L:3,D:512,M:22,EncDec}_{\text{S}}}$ | — | 0.007 72 | 0.000 56 | 1.633 85 | 110.777 |
| | FNO$_{\text{L:2,D:192,M:18,EncDec}_{\text{S}}}$ | — | 0.535 40 | 0.002 07 | 1.231 71 | 5.788 02 |
| | FNO$_{\text{L:1,D:64,M:8,EncDec}_{\text{B2}}}$ | 1.009 42 | 0.396 67 | 0.010 02 | 1.188 21 | 1.362 39 |
| | FNO$_{\text{L:1,D:64,M:8,EncDec}_{\text{B1}}}$ | 2.796 91 | 0.802 50 | 0.013 56 | 1.155 19 | 1.204 75 |
| | FNO$_{\text{L:1,D:64,M:8,EncDec}_{\text{S}}}$ | 1.436 78 | 1.359 91 | 0.015 47 | 0.852 05 | 0.953 89 |
| | FNO$_{\text{L:1,D:64,M:8,EncDec}_{\text{S}}}$ | 1.436 78 | 1.359 91 | 0.015 47 | 0.850 94 | 0.948 33 |
| depth | MP-PDE$_{\text{L:9,D:640,EncDec}_{\text{B1}}}$ | 1.389 04 | 0.765 55 | 0.013 50 | 9.259 38 | 848.036 |
| | MP-PDE$_{\text{L:7,D:640,EncDec}_{\text{B1}}}$ | 1.683 27 | 0.568 63 | 0.014 28 | 7.488 72 | 660.928 |
| | MP-PDE$_{\text{L:5,D:128,EncDec}_{\text{B2}}}$ | 0.163 93 | 0.166 24 | 0.016 52 | 5.828 85 | 30.3772 |
| | MP-PDE$_{\text{L:5,D:128,EncDec}_{\text{B1}}}$ | 0.450 55 | 0.334 69 | 0.013 19 | 5.711 71 | 30.1035 |
| | MP-PDE$_{\text{L:3,D:64,EncDec}_{\text{B2}}}$ | 4.182 10 | 1.192 08 | 0.015 60 | 3.528 91 | 8.326 50 |
| | MP-PDE$_{\text{L:3,D:64,EncDec}_{\text{B1}}}$ | 2.089 42 | 1.409 94 | 0.015 80 | 3.431 75 | 8.162 14 |
| | MP-PDE$_{\text{L:1,D:64,EncDec}_{\text{B2}}}$ | — | 0.965 22 | 0.013 89 | 1.981 64 | 3.537 74 |
| | MP-PDE$_{\text{L:1,D:64,EncDec}_{\text{B1}}}$ | 1.952 42 | 1.335 02 | 0.017 36 | 1.890 47 | 3.363 27 |
| | MP-PDE$_{\text{L:1,D:64,EncDec}_{\text{S}}}$ | 2.206 77 | 1.326 38 | 0.023 48 | 1.607 72 | 3.082 30 |
| | MP-PDE$_{\text{L:1,D:64,EncDec}_{\text{S}}}$ | 2.206 76 | 1.326 57 | 0.023 50 | 1.576 80 | 3.089 18 |
| width | MP-PDE$_{\text{L:7,D:2432,EncDec}_{\text{B1}}}$ | 0.054 16 | 0.084 21 | 0.012 63 | 9.405 97 | 8866.54 |
| | MP-PDE$_{\text{L:5,D:2432,EncDec}_{\text{B1}}}$ | 0.096 11 | 0.134 01 | 0.013 41 | 7.135 19 | 6334.37 |
| | MP-PDE$_{\text{L:5,D:1536,EncDec}_{\text{B2}}}$ | 0.044 48 | 0.003 76 | 0.003 79 | 5.856 46 | 2575.59 |
| | MP-PDE$_{\text{L:5,D:1536,EncDec}_{\text{B1}}}$ | 0.021 80 | 0.040 98 | 0.012 31 | 5.699 30 | 2573.31 |
| | MP-PDE$_{\text{L:2,D:1920,EncDec}_{\text{B2}}}$ | 0.001 09 | 0.001 15 | 0.000 62 | 3.299 61 | 1623.96 |
| | MP-PDE$_{\text{L:2,D:1920,EncDec}_{\text{B1}}}$ | 0.204 60 | 0.242 57 | 0.012 41 | 3.183 53 | 1622.10 |
| | MP-PDE$_{\text{L:1,D:128,EncDec}_{\text{B2}}}$ | 2.294 86 | 0.102 77 | 0.015 34 | 2.191 44 | 7.271 26 |
| | MP-PDE$_{\text{L:1,D:128,EncDec}_{\text{B1}}}$ | 1.635 61 | 1.051 39 | 0.015 95 | 2.05083 | 6.938 01 |
| | MP-PDE$_{\text{L:1,D:128,EncDec}_{\text{S}}}$ | 2.293 74 | 1.350 59 | 0.021 95 | 1.755 42 | 6.739 53 |
| | MP-PDE$_{\text{L:1,D:128,EncDec}_{\text{S}}}$ | 2.293 85 | 1.350 36 | 0.021 94 | 1.754 85 | 6.736 66 |
| depth | FT$_{\text{L:9,D:512,MLP:640,NH:3,EncDec}_{\text{B1}}}$ | 0.002 64 | 0.001 14 | 0.000 23 | 9.248 77 | 158.036 |
| | FT$_{\text{L:7,D:256,MLP:512,NH:2,EncDec}_{\text{B2}}}$ | 0.006 34 | 0.004 12 | 0.000 46 | 7.039 63 | 46.7115 |
| | FT$_{\text{L:7,D:256,MLP:512,NH:2,EncDec}_{\text{B1}}}$ | 0.004 68 | 0.003 26 | 0.000 37 | 6.894 47 | 46.1182 |
| | FT$_{\text{L:6,D:128,MLP:512,NH:2,EncDec}_{\text{B2}}}$ | 0.011 20 | 0.007 04 | 0.000 69 | 5.444 32 | 19.9295 |
| | FT$_{\text{L:6,D:128,MLP:512,NH:2,EncDec}_{\text{B1}}}$ | 0.027 55 | 0.012 42 | 0.000 98 | 5.291 66 | 19.5117 |
| | FT$_{\text{L:4,D:64,MLP:320,NH:2,EncDec}_{\text{B1}}}$ | 0.702 94 | 0.047 05 | 0.002 63 | 3.546 36 | 7.397 04 |
| | FT$_{\text{L:1,D:128,MLP:192,NH:2,EncDec}_{\text{B2}}}$ | 0.681 01 | 0.124 18 | 0.005 21 | 1.785 45 | 3.771 28 |
| | FT$_{\text{L:1,D:128,MLP:192,NH:2,EncDec}_{\text{B1}}}$ | 0.134 45 | 0.110 40 | 0.005 53 | 1.648 35 | 3.42118 |
| | FT$_{\text{L:1,D:448,MLP:192,NH:3,EncDec}_{\text{S}}}$ | — | 0.124 45 | 0.001 93 | 1.628 12 | 12.4719 |
| | FT$_{\text{L:1,D:64,MLP:64,NH:1,EncDec}_{\text{B1}}}$ | 0.425 59 | 0.320 91 | 0.019 31 | 1.447 3165 | 1.943 80 |
| | FT$_{\text{L:1,D:64,MLP:64,NH:2,EncDec}_{\text{S}}}$ | 3.632 96 | 0.179 05 | 0.009 50 | 1.262 3131 | 1.983 13 |
| | FT$_{\text{L:1,D:64,MLP:64,NH:1,EncDec}_{\text{S}}}$ | 1.156 85 | 0.674 46 | 0.014 13 | 1.165 31 | 1.665 28 |
| width | FT$_{\text{L:7,D:3840,MLP:4096,NH:6,EncDec}_{\text{B1}}}$ | — | — | — | 9.182 13 | 5513.29 |
| | FT$_{\text{L:7,D:3328,MLP:3840,NH:5,EncDec}_{\text{B1}}}$ | 2.605 87 | 1.676 81 | 0.032 31 | 7.588 66 | 4234.88 |
| | FT$_{\text{L:5,D:1792,MLP:2048,NH:4,EncDec}_{\text{B2}}}$ | 2.640 14 | 1.677 62 | 0.032 31 | 5.866 95 | 910.374 |
| | FT$_{\text{L:5,D:1792,MLP:2048,NH:4,EncDec}_{\text{B1}}}$ | 0.000 98 | 0.000 75 | 0.000 22 | 5.693 77 | 906.339 |
| | FT$_{\text{L:3,D:256,MLP:1280,NH:3,EncDec}_{\text{B2}}}$ | 0.028 08 | 0.008 26 | 0.000 61 | 3.834 84 | 32.3728 |
| | FT$_{\text{L:3,D:256,MLP:1280,NH:3,EncDec}_{\text{B1}}}$ | 0.113 71 | 0.012 87 | 0.000 70 | 3.711 92 | 31.7293 |
| | FT$_{\text{L:1,D:256,MLP:256,NH:1,EncDec}_{\text{B2}}}$ | 0.347 87 | 0.065 18 | 0.012 29 | 1.929 79 | 7.079 17 |
| | FT$_{\text{L:1,D:256,MLP:256,NH:1,EncDec}_{\text{B1}}}$ | 0.033 65 | .032 64 | 0.005 54 | 1.793 23 | 6.435 97 |
| | FT$_{\text{L:1,D:256,MLP:256,NH:1,EncDec}_{\text{B1}}}$ | 0.053 31 | 0.034 98 | 0.007 00 | 1.785 49 | 6.438 03 |
| | FT$_{\text{L:1,D:2816,MLP:3840,NH:2,EncDec}_{\text{S}}}$ | 0.084 61 | 0.010 72 | 0.000 75 | 1.534 36 | 458.610 |
| | FT$_{\text{L:1,D:256,MLP:256,NH:1,EncDec}_{\text{S}}}$ | 1.682 40 | 0.667 21 | 0.006 73 | 1.460 35 | 5.852 90 |

(Continued.)

**Table C1.** (Continued.)

| | | Error metrics | | | Compute time (ms) (per 2 ms) | |
|---|---|---|---|---|---|---|
| | Model$_{\mathrm{configuration}}$ | MSE | MSE$_{t<500}$ | MSE$_{\mathrm{single}}$ | GPU | CPU$_{\mathrm{1\text{-}core}}$ |
| depth | FT-FNO$_{\mathrm{L_{FT}:10,L_{FNO}:3,D:512,MLP:960,M:16,EncDec_{B2}}}$ | 1.881 62 | 1.680 76 | 0.031 25 | 9.175 74 | 275.703 |
| | FT-FNO$_{\mathrm{L_{FT}:10,L_{FNO}:3,D:512,MLP:960,M:16,EncDec_{B1}}}$ | 0.002 11 | 0.001 34 | 0.000 33 | 9.050 03 | 275.295 |
| | FT-FNO$_{\mathrm{L_{FT}:7,L_{FNO}:3,D:512,MLP:832,M:16,EncDec_{B1}}}$ | 1.489 46 | 0.998 74 | 0.023 65 | 7.009 12 | 209.876 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:3,D:352,MLP:704,M:16,EncDec_{B2}}}$ | 0.004 90 | 0.002 59 | 0.000 32 | 5.716 05 | 86.1326 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:3,D:352,MLP:704,M:16,EncDec_{B1}}}$ | 0.004 49 | 0.003 67 | 0.000 66 | 5.613 49 | 86.0099 |
| | FT-FNO$_{\mathrm{L_{FT}:2,L_{FNO}:3,D:512,MLP:896,M:15,EncDec_{B2}}}$ | 0.013 44 | 0.003 45 | 0.000 37 | 3.645 07 | 118.348 |
| | FT-FNO$_{\mathrm{L_{FT}:2,L_{FNO}:3,D:512,MLP:896,M:15,EncDec_{B1}}}$ | 0.015 97 | 0.003 72 | 0.000 48 | 3.517 48 | 118.280 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:64,MLP:64,M:9,EncDec_{B1}}}$ | — | 0.337 48 | 0.010 43 | 1.794 79 | 2.341 46 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:2,D:32,MLP:64,M:8,EncDec_{S}}}$ | — | 0.991 86 | 0.008 16 | 1.725 89 | 1.843 10 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:32,MLP:64,M:8,EncDec_{B1}}}$ | 1.771 79 | 0.861 57 | 0.015 56 | 1.669 73 | 1.951 49 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:32,MLP:64,M:8,EncDec_{S}}}$ | — | 1.269 27 | 0.010 05 | 1.455 51 | 1.59881 |
| width | FT-FNO$_{\mathrm{L_{FT}:6,L_{FNO}:2,D:896,MLP:1792,M:40,EncDec_{B2}}}$ | 2.612 45 | 1.677 38 | 0.032 31 | 9.671 48 | 713.672 |
| | FT-FNO$_{\mathrm{L_{FT}:6,L_{FNO}:2,D:896,MLP:1792,M:40,EncDec_{B1}}}$ | 0.003 63 | 0.001 32 | 0.000 38 | 9.547 98 | 710.717 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:2,D:832,MLP:1664,M:34,EncDec_{B2}}}$ | 0.003 47 | 0.002 14 | 0.000 30 | 7.657 55 | 524.350 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:2,D:832,MLP:1664,M:34,EncDec_{B1}}}$ | 0.008 43 | 0.002 09 | 0.000 64 | 7.592 06 | 521.618 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:2,D:576,MLP:1408,M:26,EncDec_{B2}}}$ | 0.004 64 | 0.002 04 | 0.000 36 | 5.656 82 | 240.784 |
| | FT-FNO$_{\mathrm{L_{FT}:5,L_{FNO}:2,D:576,MLP:1408,M:26,EncDec_{B1}}}$ | 0.009 66 | 0.001 62 | 0.000 43 | 5.600 30 | 239.314 |
| | FT-FNO$_{\mathrm{L_{FT}:4,L_{FNO}:2,D:64,MLP:256,M:8,EncDec_{B1}}}$ | 0.556 85 | 0.323 76 | 0.008 70 | 3.644 01 | 6.652 31 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:192,MLP:640,M:10,EncDec_{S}}}$ | — | 0.427 49 | 0.003 15 | 1.815 80 | 7.058 01 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:64,MLP:128,M:8,EncDec_{B1}}}$ | 3.655 52 | 0.572 66 | 0.010 47 | 1.773 69 | 2.381 60 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:64,MLP:128,M:8,EncDec_{B1}}}$ | 2.071 86 | 1.249 12 | 0.009 86 | 1.771 89 | 2.380 57 |
| | FT-FNO$_{\mathrm{L_{FT}:1,L_{FNO}:1,D:64,MLP:128,M:8,EncDec_{S}}}$ | — | 1.407 75 | 0.006 63 | 1.475 42 | 2.087 89 |

timesteps in the block, flattened and mapped to an embedding of 4 dimensions, with no hidden layers and GELU activations.

**B1** is used for all model sizes, whereas **B2** is used to re-evaluate the six best configurations of each model. We evaluate **B2** because convolving directly over hidden features in the decoder as done in **B1** can lead to unintentional sparsity when $D \gg w$ since the kernel will not slide over each hidden feature; in **B2** a learned downsampling is added. In practice we did not find this artifact to have a significant impact in the majority of configurations. **S** is used in the smallest configurations to avoid the encoder/decoder components being the bottleneck for the inference cost.

In all processor architectures GELU activations [82] are used where applicable. The hyperparameters for each specific architecture are as follows:

- **DRN.** We use convolutional layers with kernel size 5 and the dilation pattern from [16]: Dilation rates of (1, 2, 4, 8, 4, 2, 1). For the smallest models (with EncDec$_{\mathrm{S}}$) we use a kernel size of 3. We vary the number of blocks $L$ and the number of hidden dimensions $D$.
- **UNet.** We use the modern UNet implementation from [14]. Each residual block consists of two convolutional layers with kernel size 3, a shortcut connection and group normalization [83]. At the most subsampled layer, self attention [44] is used. We vary the number of hidden dimensions $D$, alongside the number of downsampling (and corresponding upsampling) steps, denoted with *Depth*. The number of hidden dimensions $D$ is fixed throughout the network.
- **FNO.** We use the implementation from [15], and vary the number of blocks $L$, the number of hidden dimensions $D$, and the number of fourier modes $M$.
- **MP-PDE.** We use the implementation from [12], and connect each node to its 4 nearest neighbors (corresponding to a convolution of kernel size 5). We vary the number of blocks $L$ and the number of hidden dimensions $D$.
- **FT.** We use the 'transformer with fourier-type attention' implementation from [13] and repeat a set of transformer encoder blocks. We vary the number of blocks $L$, the number of hidden dimensions $D$, the number of dimensions in the MLP following self attention denoted as *MLP*, and the number of attention heads *NH*[11].
- **FT-FNO.** We use a series of $L_{FT}$ FT layers followed by a series of $L_{FNO}$ FNO layers, with the remaining applicable hyperparameters as described in the corresponding methods (dimensions $D$; fully connected size *MLP*; fourier modes *M*).

In table C1 the results for all models on the density ramp data are given. The table denotes the model configuration, the MSE on full test simulations (figure 8(*a*)), the MSE on the first 500 steps (figure 8(*b*)), the MSE on single-block predictions (figure 8(*c*)), and the compute cost (in milliseconds) of 1

[11] Since attention heads only split the total number of hidden dimensions in our case, the effect on inference time of varying *NH* is rather small; we scale it as larger models often also use more attention heads.

**Table C2.** The trade-off between error and speed when decreasing DIV1D's spatial grid size for density ramps. Note that all grids are non-equidistant following the scaling from [9], only the NN surrogates use equidistant grids. For both DIV1D and DIV1D$_{\text{fast}}$ the error measurements are taken with respect to their own reference solutions at 500 gridpoints. Cells are colored using the same scale as table C1.

| | | Compute time (ms) (per 2 ms) |
|---|---|---|
| Setting | MSE | CPU$_{\text{1-core}}$ |
| DIV1D-Nx500 | 0.000 00 | 642 882 |
| DIV1D-Nx450 | 0.000 94 | 422 383 |
| DIV1D-Nx400 | 0.004 91 | 361 780 |
| DIV1D-Nx300 | 0.039 63 | 301 293 |
| DIV1D-Nx200 | 0.226 77 | 89 738.1 |
| DIV1D-Nx100 | 2.595 51 | 9861.76 |
| DIV1D$_{\text{fast}}$-Nx500 | 0.000 00 | 22 839.0 |
| DIV1D$_{\text{fast}}$-Nx450 | 0.000 98 | 155 65.1 |
| DIV1D$_{\text{fast}}$-Nx400 | 0.005 34 | 8788.88 |
| DIV1D$_{\text{fast}}$-Nx300 | 0.044 17 | 4508.63 |
| DIV1D$_{\text{fast}}$-Nx200 | 0.281 26 | 8664.64 |
| DIV1D$_{\text{fast}}$-Nx100 | 2.750 22 | 415.797 |

block of 2 ms on the GPU (NVIDIA A100 40GB; figure 7(*b*)) and the CPU (AMD Rome 7H12, 1 core; figure 7(*a*)). As a comparison, the DIV1D results when scaling the internal grid are provided in table C2.

## Appendix D. Tables: fast transients results

In this appendix extra results for the fast transient data evaluation are provided. We provide the scaling of DIV1D's performance with respect to its internal spatial grid to contextualize the error quantification of DIV1D-NN. Additionally, we provide the confusion matrix for attachment/detachment predictions.

As a reference for DIV1D-NN's MSE, we provide DIV1D's MSE as it scales when decreasing the internal grid. In this setting, DIV1D$_{\text{fast}}$ was used to generate the dataset. The scaling of DIV1D$_{\text{fast}}$ is provided in table D1. For the density ramp data the best NN surrogates were comparable in MSE to running DIV1D at ≈450 gridpoints, whereas for the fast transient data the NN surrogate had an error residing somewhere between running DIV1D$_{\text{fast}}$ at 300 and 400 gridpoints (0.022 98 for DIV1D-NN, compared to 0.067 75 and 0.015 69 for DIV1D$_{\text{fast}}$ using 300 and 400 gridpoints, respectively).

For predicting whether the plasma is in an attached or detached state (defined as the target temperature being above or below 7 eV, see also section 5.4), DIV1D-NN's accuracy with respect to DIV1D's reference solutions is 98.61%. For more detail, the confusion matrix is provided in table D2.

**Table D1.** The trade-off between error and speed when decreasing DIV1D's spatial grid size for fast transients. The error measurements are taken with respect to reference solutions at 500 gridpoints. Error cells are colored such that green indicates better results, scaled to the DIV1D-NN error on this dataset. Compute time cells are colored with the same scale as in table C2.

| | | Compute time (ms) (per 2 ms) |
|---|---|---|
| Setting | MSE | CPU$_{\text{1-core}}$ |
| DIV1D$_{\text{fast}}$-Nx500 | 0.000 00 | 661 04.1 |
| DIV1D$_{\text{fast}}$-Nx450 | 0.006 48 | 455 20.9 |
| DIV1D$_{\text{fast}}$-Nx400 | 0.015 69 | 306 00.1 |
| DIV1D$_{\text{fast}}$-Nx300 | 0.067 75 | 134 03.2 |
| DIV1D$_{\text{fast}}$-Nx200 | 0.198 63 | 129 64.4 |
| DIV1D$_{\text{fast}}$-Nx100 | 0.363 71 | 7259.12 |

**Table D2.** Confusion matrix of attachment predictions for the fast transient data, where attachment is defined as the target temperature being below 7 eV. The accuracy of predicting attachment is 98.61% over the entire test set.

| | | DIV1D$_{\text{fast}}$ | |
|---|---|---|---|
| | | Detached | Attached |
| DIV1D-NN | Detached | 325 565 | 2123 |
| | Attached | 4170 | 120 142 |

## Appendix E. Tables: inter- and extrapolation results

In this appendix we provide more results for the inter- and extrapolation evaluation of DIV1D-NN on the density ramp data. To evaluate the ability of DIV1D-NN to interpolate and extrapolate within the parameter space, we train separate models where all simulations containing either a specific value for the upstream heat flux ($q_{||\text{u}}$) or for the impurity fraction ($\xi_{\text{C}}$) are left out from the training and validation data. These models are then tested on simulations with these parameter values, to evaluate the quality when simulating unseen parameters. In table E1, the results for all values of $q_{||\text{u}}$ and $\xi_{\text{C}}$ are given.

Additionally, we consider linear interpolation of surrounding parameter values as a baseline for this experiment. New simulations are generated by linearly inter- or extrapolating simulations of identical density ramps with the surrounding parameter values for $q_{||\text{u}}$ and $\xi_{\text{C}}$, and are then compared to the reference simulations with these parameter values. These results are provided in table E2. The ratios of errors between DIV1D-NN and linear interpolation are provided in table E3. In general, DIV1D-NN performed much better, as also demonstrated in section 5.6: The NN-based approach manages to capture non-linear dependencies in the dynamics. However, it is still not advisable to use the NN-based surrogate for extrapolation, as indicated by the higher errors (and worse ratios compared to linear interpolation) on the boundaries of the parameter space.

**Table E1.** Validating the inter- and extrapolation capabilities of DIV1D-NN. We train different models from scratch where we leave out *all* simulations using a given upstream heat flux ($q_{\|\mathrm{u}}$) *or* impurity fraction ($\xi_{\mathrm{C}}$). The values indicate the MSE with these left-out simulations, when using the remaining simulations for training and validation (cells colored by MSE, greener is better). As expected, when leaving out data within the domain (middle cells), such that we are strictly interpolating in parameter space, the model performs a lot better.

| $q_{\|\mathrm{u}}$<br>$\xi_{\mathrm{C}}$ | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| 0.01 | 0.0239 | 0.0321 | 0.0090 | 0.0073 | 0.0694 |
| 0.02 | 0.0104 | 0.0015 | 0.0011 | 0.0021 | 0.0254 |
| 0.03 | 0.0141 | 0.0020 | 0.0010 | 0.0017 | 0.0892 |
| 0.04 | 0.0074 | 0.0015 | 0.0015 | 0.0025 | 0.0130 |
| 0.05 | 0.0160 | 0.0059 | 0.0039 | 0.0048 | 0.2290 |

**Table E2.** Evaluation of the inter- and extrapolation error when using linear combinations of simulations with surrounding values to compute a given simulation. For each setting of the upstream heat flux ($q_{\|\mathrm{u}}$) and impurity fraction ($\xi_{\mathrm{C}}$), the simulations are computed by using only linear interpolation (or extrapolation) of simulations without either of those values present; cells indicate the MSE with reference simulations for the given setting. Cells are colored by MSE, using the same scale as table E1.

| $q_{\|\mathrm{u}}$<br>$\xi_{\mathrm{C}}$ | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| 0.01 | 0.0354 | 0.0105 | 0.0100 | 0.0082 | 0.0313 |
| 0.02 | 0.0313 | 0.0090 | 0.0085 | 0.0067 | 0.0224 |
| 0.03 | 0.0311 | 0.0083 | 0.0079 | 0.0061 | 0.0221 |
| 0.04 | 0.0312 | 0.0082 | 0.0077 | 0.0059 | 0.0220 |
| 0.05 | 0.0349 | 0.0086 | 0.0080 | 0.0061 | 0.0216 |

**Table E3.** Ratio of MSEs between DIV1D-NN and linear interpolation, that is, between the values from tables E1 and E2. Purple indicates the NN-based interpolation of parameter values performed better, whereas orange indicates that linear interpolation performed better; magnitudes are scaled according to the relative error. In general, NN-based interpolation results in a significantly lower error (as also demonstrated in section 5.6), however, for extrapolation it does not necessarily perform better.

| $q_{\|\mathrm{u}}$<br>$\xi_{\mathrm{C}}$ | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| 0.01 | 0.68× | 3.06× | 0.90× | 0.89× | 2.22× |
| 0.02 | 0.33× | 0.17× | 0.13× | 0.31× | 1.13× |
| 0.03 | 0.45× | 0.24× | 0.13× | 0.28× | 4.04× |
| 0.04 | 0.24× | 0.18× | 0.19× | 0.42× | 0.59× |
| 0.05 | 0.46× | 0.69× | 0.49× | 0.79× | 10.60× |

## ORCID iDs

Yoeri Poels https://orcid.org/0000-0002-4071-4855
Gijs Derks https://orcid.org/0000-0003-3420-0388
Egbert Westerhof https://orcid.org/0000-0002-0749-9399
Koen Minartz https://orcid.org/0000-0002-6459-8692
Sven Wiesen https://orcid.org/0000-0002-3696-5475
Vlado Menkovski https://orcid.org/0000-0001-5262-0605

## References

[1] Kukushkin A., Pacher H., Kotov V., Pacher G. and Reiter D. 2011 Finalizing the ITER divertor design: the key role of SOLPS modeling *Fusion Eng. Des.* **86** 2865–73

[2] Wiesen S., Groth M., Wischmeier M., Brezinsek S., Jarvinen A., Reimold F. and Aho-Mantila L. (JET contributors, EUROfusion MST1 team, ASDEX Upgrade team and Alcator C-mod team) 2017 Plasma edge and plasma-wall interaction modelling: Lessons learned from metallic devices *Nucl. Mater. Energy* **12** 3–17

[3] Wischmeier M. (ASDEX Upgrade team and JET EFDA contributors) 2015 High density operation for reactor-relevant power exhaust *J. Nucl. Mater.* **463** 22–29

[4] Pacher H., Kukushkin A., Pacher G., Kotov V., Pitts R. and Reiter D. 2015 Impurity seeding in ITER DT plasmas in a carbon-free environment *J. Nucl. Mater.* **463** 591–5

[5] Pitts R. *et al* 2019 Physics basis for the first ITER tungsten divertor *Nucl. Mater. Energy* **20** 100696

[6] Wiesen S. *et al* 2015 The new SOLPS-ITER code package *J. Nucl. Mater.* **463** 480–4

[7] Rognlien T.D., Brown P.N., Campbell R.B., Kaiser T.B., Knoll D.A., McHugh P.R., Porter G.D., Rensink M.E. and Smith G.R. 1994 2-D fluid transport simulations of gaseous/radiative divertors *Contrib. Plasma Phys.* **34** 362–7

[8] Dudson B.D., Allen J., Body T., Chapman B., Lau C., Townley L., Moulton D., Harrison J. and Lipschultz B. 2019 The role of particle, energy and momentum losses in 1D simulations of divertor detachment *Plasma Phys. Control. Fusion* **61** 065008

[9] Derks G.L., Frankemölle J.P.K.W., Koenders J.T.W., van Berkel M., Reimerdes H., Wensing M. and Westerhof E. 2022 Benchmark of a self-consistent dynamic 1D divertor model DIV1d using the 2D SOLPS-ITER code *Plasma Phys. Control. Fusion* **64** 125013

[10] Stangeby P. and Moulton D. 2020 A simple analytic model of impurity leakage from the divertor and accumulation in the main scrape-off layer *Nucl. Fusion* **60** 106005

[11] Siccinio M. *et al* 2022 Impact of the plasma operation on the technical requirements in EU-DEMO *Fusion Eng. Des.* **179** 113123

[12] Brandstetter J., Worrall D.E. and Welling M. 2022 Message passing neural PDE solvers *Int. Conf. on Learning Representations* vol 10 (available at: https://openreview.net/forum?id=vSix3HPYKSU).

[13] Cao S. 2021 Choose a transformer: Fourier or galerkin *Advances in Neural Information Processing Systems* (*6–14 December 2021*) vol 34 pp 24924–40 (available at: https://proceedings.neurips.cc/paper/2021/hash/d0921d442ee91b896ad95059d13df618-Abstract.html)

[14] Gupta J.K. and Brandstetter J. 2022 Towards multi-spatiotemporal-scale generalized PDE modeling (arXiv:2209.15616)

[15] Li Z., Kovachki N.B., Azizzadenesheli K., Liu B., Bhattacharya K., Stuart A.M. and Anandkumar A. 2021 Fourier neural operator for parametric partial differential equations *Int. Conf. on Learning Representations* (*3–7 May 2021*) vol 9 (available at: https://openreview.net/forum?id=c8P9NQVtmnO).

[16] Stachenfeld K., Fielding D.B., Kochkov D., Cranmer M., Pfaff T., Godwin J., Cui C., Ho S., Battaglia P. and Sanchez-Gonzalez A. 2022 Learned simulators for turbulence *Int. Conf. on Learning Representations* (*25–29 April 2022*) vol 10 (available at: https://openreview.net/forum?id=msRBojTz-Nh)

[17] Ravensbergen T. *et al* 2021 Real-time feedback control of the impurity emission front in tokamak divertor plasmas *Nat. Commun.* **12** 1105

[18] Capes H., Ghendrih P. and Samain A. 1992 Radiative instability in a diverted plasma *Phys. Fluids* B **4** 1287–93
[19] Loarte A. *et al* 1998 Plasma detachment in JET Mark I divertor experiments *Nucl. Fusion* **38** 331–71
[20] Zohm H. 1996 Edge localized modes (ELMs) *Plasma Phys. Control. Fusion* **38** 105–28
[21] Pau A., Fanni A., Carcangiu S., Cannas B., Sias G., Murari A. and Rimini F. (JET Contributors) 2019 A machine learning approach based on generative topographic mapping for disruption prevention and avoidance at JET *Nucl. Fusion* **59** 106017
[22] Zhu J., Rea C., Montes K., Granetz R., Sweeney R. and Tinguely R. 2020 Hybrid deep-learning architecture for general disruption prediction across multiple tokamaks *Nucl. Fusion* **61** 026007
[23] Fischer R., Fuchs C.J., Kurzan B., Suttrop W. and Wolfrum E. (ASDEX Upgrade Team) 2010 Integrated data analysis of profile diagnostics at ASDEX upgrade *Fusion Sci. Technol.* **58** 675–84
[24] Pavone A., Svensson J., Kwak S., Brix M. and Wolf R.C. (JET Contributors) 2020 Neural network approximated Bayesian inference of edge electron density profiles at JET *Plasma Phys. Control. Fusion* **62** 045019
[25] Ho A., Citrin J., Bourdelle C., Camenen Y., Casson F.J., van de Plassche K.L. and Weisen H. (JET Contributors) 2021 Neural network surrogate of QuaLiKiz using JET experimental data to populate training space *Phys. Plasmas* **28** 032305
[26] Meneghini O. *et al* 2020 Neural-network accelerated coupled core-pedestal simulations with self-consistent transport of impurities and compatible with ITER IMAS *Nucl. Fusion* **61** 026006
[27] Anirudh R. *et al* 2022 2022 review of data-driven plasma science (arXiv:2205.15832)
[28] Dasbach S. and Wiesen S. 2023 Towards fast surrogate models for interpolation of tokamak edge plasmas *Nucl. Mater. Energy* **34** 101396
[29] Gopakumar V. and Samaddar D. 2020 Image mapping the temporal evolution of edge characteristics in tokamaks using neural networks *Mach. Learn.: Sci. Technol.* **1** 015006
[30] Lore J., Pascuale S.D., Laiu P., Russo B., Park J.-S., Park J., Brunton S., Kutz J. and Kaptanoglu A. 2023 Time-dependent SOLPS-ITER simulations of the tokamak plasma boundary for model predictive control using SINDy *Nucl. Fusion* **63** 046015
[31] Brunton S.L., Proctor J.L. and Kutz J.N. 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems *Proc. Natl Acad. Sci.* **113** 3932–7
[32] Gopakumar V., Pamela S., Zanisi L., Li Z., Anandkumar A. and (MAST Team) 2023 Fourier neural operator for plasma modelling (arXiv:2302.06542)
[33] Zhu B., Zhao M., Bhatia H., qiao Xu X., Bremer P.-T., Meyer W., Li N. and Rognlien T. 2022 Data-driven model for divertor plasma detachment prediction *J. Plasma Phys.* **88** 895880504
[34] Chen T.Q., Rubanova Y., Bettencourt J. and Duvenaud D. 2018 Neural ordinary differential equations *Advances in Neural Information Processing Systems* (*Montréal, Canada*, *2–8 December 2018*) vol 31 pp 6572–83 (available at: https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html)
[35] Brown P.N., Byrne G.D. and Hindmarsh A.C. 1989 VODE: a variable-coefficient ODE solver *SIAM J. Sci. Stat. Comput.* **10** 1038–51
[36] Byrne G. and Thompson S. 2013 DVODE_F90: a variable-coefficient ODE solver (available at: www.radford.edu/thompson/vodef90web/)
[37] Shannon C. 1949 Communication in the presence of noise *Proc. IRE* **37** 10–21
[38] Frerichs H., Bonnin X., Feng Y., Li L., Liu Y., Loarte A., Pitts R., Reiter D. and Schmitz O. 2021 Divertor detachment in the pre-fusion power operation phase in ITER during application of resonant magnetic perturbations *Nucl. Fusion* **61** 126027
[39] LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W. and Jackel L.D. 1989 Backpropagation applied to handwritten zip code recognition *Neural Comput.* **1** 541–51
[40] He K., Zhang X., Ren S. and Sun J. 2016 Deep residual learning for image recognition *Conf. on Computer Vision and Pattern Recognition* (*Las Vegas, NV, USA*, *27–30 June 2016*) (IEEE) https://doi.org/10.1109/cvpr.2016
[41] Yu F. and Koltun V. 2016 Multi-scale context aggregation by dilated convolutions *Int. Conf. on Learning Representations* (*San Juan, Puerto Rico*, *2–4 May 2016*) vol 4 http://doi.org/10.48550/arXiv.1511.07122
[42] Ronneberger O., Fischer P. and Brox T. 2015 U-net: Convolutional networks for biomedical image segmentation *Medical Image Computing and Computer-Assisted Intervention* (Springer) pp 234–41 https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28
[43] Ho J., Jain A. and Abbeel P. 2020 Denoising diffusion probabilistic models *Advances in Neural Information Processing Systems* (*6–12 December 2020*) vol 33 pp 6840–51 (available at: https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html)
[44] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L.U. and Polosukhin I. 2017 Attention is all you need *Advances in Neural Information Processing Systems* (*Long Beach, CA, USA*, *4–9 December 2017*) vol 30 (available at: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html)
[45] Cooley J.W. and Tukey J.W. 1965 An algorithm for the machine calculation of complex fourier series *Math. Comput.* **19** 297–301
[46] Gilmer J., Schoenholz S.S., Riley P.F., Vinyals O. and Dahl G.E. 2017 Neural message passing for quantum chemistry *Proc. 34th Int. Conf. on Machine Learning (Proc. Machine Learning Research) (PMLR)* (*Sydney, Australia*, *6–11 August 2017*) vol 70 pp 1263–72 (available at: https://proceedings.mlr.press/v70/gilmer17a.html)
[47] Ba J.L., Kiros J.R., and Hinton G.E. 2016 Layer normalization (arXiv:1607.06450)
[48] Kingma D.P. and Ba J. 2015 Adam: a method for stochastic optimization *Int. Conf. on Learning Representations* (*San Diego, CA, USA*, *7–9 May 2015*) vol 3 http://doi.org/10.48550/arXiv.1412.6980
[49] Paszke A. *et al* 2019 Pytorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* (*Vancouver, Canada*, *8–14 December 2019*) vol 32 (available at: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html)
[50] NVIDIA 2023 TensorRT SDK (available at: https://developer.nvidia.com/tensorrt)
[51] Post D., Jensen R., Tarter C., Grasberger W. and Lokke W. 1977 Steady-state radiative cooling rates for low-density, high-temperature plasmas *At. Data Nucl. Data Tables* **20** 397–439
[52] Koenders J., Perek A., Kool B., Février O., Ravensbergen T., Galperti C., Duval B., Theiler C. and van Berkel M. 2022

Model-based impurity emission front control using deuterium fueling and nitrogen seeding in TCV *Nucl. Fusion* **63** 026006

[53] Smolders A. *et al* (the TCV team) 2020 Comparison of high density and nitrogen seeded detachment using SOLPS-ITER simulations of the tokamak á configuration variable *Plasma Phys. Control. Fusion* **62** 125006

[54] Theiler C. *et al* 2017 Results from recent detachment experiments in alternative divertor configurations on TCV *Nucl. Fusion* **57** 072008

[55] Stangeby P.C. 2018 Basic physical processes and reduced models for plasma detachment *Plasma Phys. Control. Fusion* **60** 044022

[56] Settles B. 2009 Active learning literature survey *Computer Sciences Technical Report 1648* (University of Wisconsin–Madison)

[57] Fable E. *et al* 2021 The modeling of a tokamak plasma discharge, from first principles to a flight simulator *Plasma Phys. Control. Fusion* **64** 044002

[58] Felici F., Citrin J., Teplukhina A., Redondo J., Bourdelle C., Imbeaux F. and Sauter O. 2018 Real-time-capable prediction of temperature and density profiles in a tokamak using RAPTOR and a first-principle-based transport model *Nucl. Fusion* **58** 096006

[59] Romanelli M. *et al* 2014 JINTRAC: A system of codes for integrated simulation of tokamak scenarios *Plasma Fusion Res.* **9** 3403023

[60] Bosman T., van Berkel M. and de Baar M. 2022 Constrained model-predictive control of the electron density profile in ITER using two pellet injectors *2022 IEEE Conf. on Control Technology and Applications (CCTA)* (*Trieste, Italy*, *22–25 August 2022*) (IEEE) https://doi.org/10.1109/ccta49430.2022.9966088

[61] Gin C., Lusch B., Brunton S.L. and Kutz J.N. 2020 Deep learning models for global coordinate transformations that linearise PDEs *Eur. J. Appl. Math.* **32** 515–39

[62] Lusch B., Kutz J.N. and Brunton S.L. 2018 Deep learning for universal linear embeddings of nonlinear dynamics *Nat. Commun.* **9** 4950

[63] Kalman R.E. 1960 A new approach to linear filtering and prediction problems *J. Basic Eng.* **82** 35–45

[64] Biel W. *et al* 2019 Diagnostics for plasma control—from ITER to DEMO *Fusion Eng. Des.* **146** 465–72

[65] Rebain D., Matthews M.J., Yi K.M., Sharma G., Lagun D. and Tagliasacchi A. 2022 Attention beats concatenation for conditioning neural fields (arXiv:2209.10684)

[66] Ruhe D., Gupta J.K., de Keninck S., Welling M. and Brandstetter J. 2023 Geometric clifford algebra networks (arXiv:2302.06594)

[67] Karniadakis G.E., Kevrekidis I.G., Lu L., Perdikaris P., Wang S. and Yang L. 2021 Physics-informed machine learning *Nat. Rev. Phys.* **3** 422–40

[68] Richter-Powell J., Lipman Y. and Chen R.T.Q. 2022 Neural conservation laws: a divergence-free perspective *Advances in Neural Information Processing Systems* (*New Orleans, LA, USA*, *28 November–29 December 2022* ) (available at: https://openreview.net/forum?id=prQkA_NjuuB)

[69] Kochkov D., Smith J.A., Alieva A., Wang Q., Brenner M.P. and Hoyer S. 2021 Machine learning–accelerated computational fluid dynamics *Proc. Natl Acad. Sci.* **118** e2101784118

[70] Lakhlili J., Hoenen O., Luk O.O. and Coster D.P. 2020 Uncertainty quantification for multiscale fusion plasma simulations with vecma toolkit *Computational Science – ICCS 2020* (Springer) pp 719–30 https://doi.org/10.1007/978-3-030-50436-6_53

[71] Casali L., Fable E., Dux R. and Ryter F. (ASDEX Upgrade team) 2018 Modelling of nitrogen seeding experiments in the ASDEX upgrade tokamak *Phys. Plasmas* **25** 032506

[72] Felici F., Sauter O., Coda S., Duval B., Goodman T., Moret J.-M. and Paley J. (the TCV Team) 2011 Real-time physics-model-based simulation of the current density profile in tokamak plasmas *Nucl. Fusion* **51** 083052

[73] Hoelzl M. *et al* 2021 The JOREK non-linear extended MHD code and applications to large-scale instabilities and their control in magnetically confined fusion plasmas *Nucl. Fusion* **61** 065001

[74] Pereverzev G. and Yushmanov P.N. 2002 Astra automated system for transport analysis in a tokamak *Technical Report IPP–5-98* (Max-Planck-Institut für Plasmaphysik) (available at: http://inis.iaea.org/search/search.aspx?orig_q=RN:33018446)

[75] Raj S., Bisai N., Shankar V. and Sen A. 2020 Effects of nitrogen seeding in a tokamak plasma *Phys. Plasmas* **27** 122302

[76] Fietz S., Fable E., Hobirk J., Fischer R., Fuchs C., Pereverzev G. and Ryter F. (the ASDEX Upgrade Team) 2013 Investigation of transport models in ASDEX upgrade current ramps *Nucl. Fusion* **53** 053004

[77] Pamela S. *et al* 2017 Recent progress in the quantitative validation of JOREK simulations of ELMs in JET *Nucl. Fusion* **57** 076006

[78] Teplukhina A.A., Sauter O., Felici F., Merle A. and Kim D. (TCV team, ASDEX Upgrade team and EUROfusion MST1 team) 2017 Simulation of profile evolution from ramp-up to ramp-down and optimization of tokamak plasma termination with the RAPTOR code *Plasma Phys. Control. Fusion* **59** 124004

[79] Wensing M. *et al* (TCV team and EUROfusion MST1 team) 2021 SOLPS-ITER validation with TCV l-mode discharges *Phys. Plasmas* **28** 082508

[80] Park J.-S., Bonnin X. and Pitts R. 2020 Assessment of ITER divertor performance during early operation phases *Nucl. Fusion* **61** 016021

[81] Ramachandran P., Zoph B. and Le Q. 2018 Searching for activation functions *Int. Conf. on Learning Representations, Workshop Track Proc.* vol 6

[82] Hendrycks D. and Gimpel K. 2016 Gaussian error linear units (gelus) (arXiv:1606.08415)

[83] Wu Y. and He K. 2019 Group normalization *Int. J. Comput. Vis.* **128** 742–55