

TORAX: A Fast and Differentiable Tokamak Transport Simulator in JAX

Jonathan Citrin,^{*} Ian Goodfellow, Akhil Raju, Jeremy Chen, Jonas Degraeve, Craig Donner, Federico Felici, Philippe Hamel, Andrea Huber, Dmitry Nikulin, David Pfau, Brendan Tracey, Martin Riedmiller, and Pushmeet Kohli
Google DeepMind, London, UK

We present TORAX^a, a new, open-source, differentiable tokamak core transport simulator implemented in Python using the JAX framework. TORAX solves the coupled equations for ion heat transport, electron heat transport, particle transport, and current diffusion, incorporating modular physics-based and ML models. JAX’s just-in-time compilation ensures fast runtimes, while its automatic differentiation capability enables gradient-based optimization workflows and simplifies the use of Jacobian-based PDE solvers. Coupling to ML-surrogates of physics models is greatly facilitated by JAX’s intrinsic support for neural network development and inference. TORAX is verified against the established RAPTOR code, demonstrating agreement in simulated plasma profiles. TORAX provides a powerful and versatile tool for accelerating research in tokamak scenario modeling, pulse design, and control.

I. INTRODUCTION

Toroidal magnetic confinement devices such as tokamaks and stellarators are currently the leading candidates for fusion energy reactors [1]. Accurate integrated plasma simulation is crucial for a wide range of applications, including interpretation of experiments, physics validation and understanding, extrapolation to future performance, scenario discovery and optimization, and controller design [2]. Integrated tokamak modelling is inherently multi-scale and multi-physics, with multiple orders of magnitude in spatiotemporal scales between relevant physics processes [3]. Therefore domain decomposition and physics reduction is key.

Core transport simulation focuses on the plasma domain with closed nested magnetic flux surfaces, where due to fast transport along field lines, the transport problem can be reduced to a set of coupled 1D transport PDEs. The bulk of fusion power and plasma current is concentrated in the plasma core. Core simulation is thus a central component of prediction of the (fusion) performance of a plasma, combining plasma equilibrium, transport, fuelling, and heating.

Multiple core transport codes have been developed, including JETTO [4], ASTRA [5], PTRANSP [6], and ETS [7]. These codes offer a range of physics fidelity in their constituent physics models, with higher fidelity models often coming at the expense of simulation speed. Simulations can take hours to days on a standard compute node for simulating several energy confinement times when using state-of-the-art reduced turbulence, neoclassical, and heating modules.

However, a range of use-cases demands both fast and accurate simulations. Crucially, differentiability is essential for gradient-based optimization and sensitivity analysis, which is typically not available in traditional transport codes. Speed is also vital for many-query problems

exploring a wide range of parameter spaces, for example for uncertainty quantification (UQ), scenario optimization, inter-shot experimental preparation, model-based controller design, and reactor design.

For optimization and control-oriented applications, dedicated codes have been developed, such as RAPTOR [8, 9], METIS [10], FENIX [11], and COTSIM [12]. To circumvent the conflicting constraints of simulation speed and accuracy, machine learned (ML) surrogates of complex physics models has emerged as a promising technique [13–20]. However, all the codes listed thus far are developed in Fortran or MATLAB, which can pose challenges for (auto)-differentiation and seamless coupling to ML-surrogate models. RAPTOR stands out as a differentiable transport code, but it is not auto-differentiable, and thus relies on manually coded Jacobian calculations, which is a hindrance for extensibility, especially to ML-surrogates which are complex nested nonlinear functions.

Therefore, existing tokamak transport codes face limitations in terms of a subset of computational speed, auto-differentiability, and integration with emerging machine learning (ML) techniques.

Addressing these limitations, we present TORAX, a new, open-source, tokamak core transport simulator implemented in Python using the JAX framework [21]. JAX provides automatic differentiation and just-in-time (JIT) compilation, enabling both fast runtimes and gradient-based algorithms, including optimization and nonlinear PDE solvers. Coupling to ML-surrogates of physics models is greatly facilitated by JAX’s intrinsic support for neural network development and inference, allowing such surrogates to be developed in the native language of the simulator. Python’s ease of use and wide adoption facilitates code extensibility, maintainability, and coupling within various workflows and to additional physics models. Finally, JAX can seamlessly execute on multiple backends including CPU and GPU, with support for vectorization and parallelization.

In this paper, we outline the TORAX model and methodology, highlighting its key initial features and capabilities. We benchmark TORAX against the estab-

^{*} citrin@google.com

^a <https://github.com/google-deepmind/torax>
<https://torax.readthedocs.io>

lished RAPTOR code, demonstrating accurate and verified performance. Following this initial release, further TORAX extensions and generalizations are planned. We conclude by outlining the development roadmap.

II. TORAX MODEL DESCRIPTION

TORAX simulates the time evolution of core plasma profiles using a coupled set of 1D transport PDEs, discretized in space and time, and solved numerically. The PDEs arise from flux-surface averaged moments of the underlying kinetic equations, and from Faraday's law [22]. Their validity relies on time-scale separation of transport and turbulent dynamics, and on kinetic effects (e.g. turbulent and neoclassical dynamics) being a small perturbation to the background equilibrium distribution functions, conditions generally satisfied in the tokamak or stellarator core.

A. Governing equations

TORAX solves coupled 1D PDEs in normalized toroidal flux coordinates, $\hat{\rho}$, with $0 \leq \hat{\rho} \leq 1$. $\hat{\rho}$ is a flux surface label, being constant on a closed surface of poloidal magnetic flux ψ . It is defined as $\hat{\rho} = \sqrt{\frac{\Phi(\psi)}{\Phi_b}}$, where $\Phi(\psi)$ is the toroidal magnetic flux enclosed by the magnetic poloidal flux surface being labelled (see figure 1), and Φ_b is the toroidal flux enclosed by the plasma core boundary, i.e. the last-closed-flux-surface.

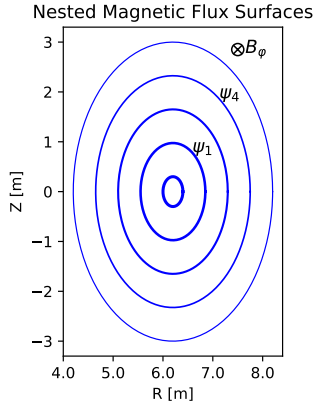


FIG. 1. Poloidal cross section of the core region of a toroidal tokamak plasma, with closed nested magnetic flux surfaces. R is the major radius from the center of the torus, Z is the vertical direction. ITER-like magnitudes are shown. The flux surfaces are elliptical in the poloidal plane for illustrative purposes, and do not correspond to a true equilibrium calculation. The toroidal magnetic field B_ϕ is in the direction into the page. ψ_1 and ψ_4 label two surfaces of constant poloidal flux at arbitrary locations. A discrete set of toroidal flux coordinates ρ is defined through integrating the toroidal magnetic field within a discrete set of poloidal flux surfaces.

The set of 1D PDEs being solved are the following:

- Ion heat transport, governing the evolution of the ion temperature T_i .

$$\frac{3}{2}V'^{-5/3} \left(\frac{\partial}{\partial t} - \frac{\dot{\Phi}_b}{2\Phi_b} \frac{\partial}{\partial \hat{\rho}} \right) [V'^{5/3} n_i T_i] = \frac{1}{V'} \frac{\partial}{\partial \hat{\rho}} \left[\chi_i n_i \frac{g_1}{V'} \frac{\partial T_i}{\partial \hat{\rho}} - g_0 q_i^{\text{conv}} T_i \right] + Q_i \quad (1)$$

- Electron heat transport, governing the evolution of the electron temperature T_e .

$$\frac{3}{2}V'^{-5/3} \left(\frac{\partial}{\partial t} - \frac{\dot{\Phi}_b}{2\Phi_b} \frac{\partial}{\partial \hat{\rho}} \right) [V'^{5/3} n_e T_e] = \frac{1}{V'} \frac{\partial}{\partial \hat{\rho}} \left[\chi_e n_e \frac{g_1}{V'} \frac{\partial T_e}{\partial \hat{\rho}} - g_0 q_e^{\text{conv}} T_e \right] + Q_e \quad (2)$$

- Electron particle transport, governing the evolution of the electron density n_e .

$$\left(\frac{\partial}{\partial t} - \frac{\dot{\Phi}_b}{2\Phi_b} \frac{\partial}{\partial \hat{\rho}} \right) [n_e V'] = \frac{\partial}{\partial \hat{\rho}} \left[D_e n_e \frac{g_1}{V'} \frac{\partial n_e}{\partial \hat{\rho}} - g_0 V_e n_e \right] + V' S_n \quad (3)$$

- Current diffusion, governing the evolution of the poloidal flux ψ .

$$\frac{16\pi^2 \sigma_{||} \mu_0 \hat{\rho} \Phi_b^2}{F^2} \left(\frac{\partial \psi}{\partial t} - \frac{\dot{\rho} \dot{\Phi}_b}{2\Phi_b} \frac{\partial \psi}{\partial \hat{\rho}} \right) = \frac{\partial}{\partial \hat{\rho}} \left(\frac{g_2 g_3}{\hat{\rho}} \frac{\partial \psi}{\partial \hat{\rho}} \right) - \frac{8\pi^2 V' \mu_0 \Phi_b}{F^2} \langle \mathbf{B} \cdot \mathbf{j}_{ni} \rangle \quad (4)$$

where $T_{i,e}$ are ion and electron temperatures, $n_{i,e}$ are ion and electron densities, and ψ is poloidal flux. $\chi_{i,e}$ are ion and electron heat conductivities, $q_{i,e}^{\text{conv}}$ are ion and electron heat convections, D_e is electron particle diffusivity, and V_e is electron particle convection. $Q_{i,e}$ are the total ion and electron heat sources, and S_n is the total electron particle source. $V' \equiv dV/d\hat{\rho}$, i.e. the flux surface volume derivative. $\sigma_{||}$ is the plasma neoclassical conductivity, and $\langle \mathbf{B} \cdot \mathbf{j}_{ni} \rangle$ is the flux-surface-averaged non-inductive current (comprised of the bootstrap current and external current drive) projected onto and multiplied by the magnetic field. $F \equiv RB_\phi$ is the toroidal field flux function, where R is major radius and B_ϕ the toroidal magnetic field. Φ_b is the toroidal flux enclosed by the last closed flux surface, and $\dot{\Phi}_b$ is its time derivative, non-zero with time-varying toroidal magnetic field and/or last closed flux surface shape. μ_0 is the permeability of free space. The geometric quantities g_0 , g_1 , g_2 and g_3 are defined as follows.

$$g_0 = \langle (\nabla V) \rangle \quad (5)$$

$$g_1 = \left\langle (\nabla V)^2 \right\rangle \quad (6)$$

where ∇V is the radial derivative of the plasma volume, and $\langle \rangle$ denotes flux surface averaging,

$$g_2 = \left\langle \frac{(\nabla V)^2}{R^2} \right\rangle \quad (7)$$

$$g_3 = \left\langle \frac{1}{R^2} \right\rangle \quad (8)$$

where R is the major radius along the flux surface being averaged.

The geometry terms V' , g_0 , g_1 , g_2 and g_3 are calculated from flux-surface-averaged outputs of a Grad-Shafranov equilibrium code (see section III), either pre-calculated or coupled to TORAX within a larger workflow.

The boundary conditions are as follows. All equations have a zero-derivative boundary condition at $\hat{\rho} = 0$. The T_i , T_e , n_e equations have fixed boundary conditions at $\hat{\rho} = 1$, which are user-defined. The ψ equation has a Neumann (derivative) boundary condition at $\hat{\rho} = 1$, which sets the total plasma current through the relation:

$$I_p = \left[\frac{\partial \psi}{\partial \rho} \frac{g_2 g_3}{\rho} \frac{R_0 J}{16\pi^4 \mu_0} \right]_{LCFS} \quad (9)$$

See Ref. [23] for a more comprehensive summary and derivation of the governing equations and associated geometric quantities. See Appendix A for a breakdown in how the Φ_b term leads to additional convective and source terms. Future work can extend the governing equations to include momentum transport, and multiple ion species including impurities. Details of the physics models underlying the PDE coefficients is provided in section III.

B. Spatial discretization scheme

TORAX employs a finite volume method (FVM) to discretize the governing PDEs in space [24]. The TORAX JAX 1D FVM package is significantly influenced by FiPy [25]. The 1D spatial domain, $0 \leq \hat{\rho} \leq 1$, is divided into a uniform grid of N cells, each with a width of $d\hat{\rho} = 1/N$. The cell centers are denoted by $\hat{\rho}_i$, where $0 = 1, 2, \dots, N-1$, and the $N+1$ cell faces are located at $\hat{\rho}_{i\pm 1/2}$. Both $\hat{\rho} = 0$ and $\hat{\rho} = 1$ are on the face grid. TORAX is currently restricted to a uniformly spaced grid, with generalization to a non-uniform grid left for future development. See figure 2 for an illustration of the spatial grid.

The FVM approach involves integrating each PDE over a control volume (in this case, a cell) and applying the divergence theorem to convert volume integrals to surface integrals. This leads to a system of algebraic equations for the cell-averaged values of the plasma profiles.

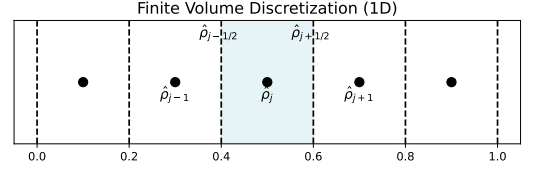


FIG. 2. 1D spatial discretization with the finite volume method, with $N = 5$ and $\hat{\rho} = 0.2$. The cell volume centered at $\hat{\rho} = 0.5$ is highlighted. Keeping with 3D FVM formalism, a cell volume is defined with $V = d\hat{\rho}A$, with arbitrary face area A . The corresponding cell faces are located at $\hat{\rho} = 0.4$ and $\hat{\rho} = 0.6$. In the 1D case spatial inhomogeneity is only along the $\hat{\rho}$ direction, and the final equations are normalized by V , cancelling out A .

For a generic conservation law of the form:

$$\frac{\partial x}{\partial t} + \nabla \cdot \mathbf{\Gamma} = S \quad (10)$$

where x is a conserved quantity, $\mathbf{\Gamma}$ is the flux, and S is a source term, the FVM discretization followed by volume integration and applying the divergence theorem for cell i yields:

$$\frac{\partial}{\partial t}(V_i x_i) + (\mathbf{\Gamma}_{i+1/2} \cdot \mathbf{A}_{i+1/2} - \mathbf{\Gamma}_{i-1/2} \cdot \mathbf{A}_{i-1/2}) = V_i S_i, \quad (11)$$

where: x_i is the cell-averaged value of x in cell i . V_i is the volume of cell i . $\mathbf{\Gamma}_{i+1/2}$ is the flux at face $i+1/2$. $\mathbf{A}_{i+1/2}$ is the area of face $i+1/2$. S_i is the cell-averaged source term in cell i .

Restricting to 1D, the face areas are a constant arbitrary A , and the volumes $V_i = d\hat{\rho}_i A$. The area cancels out when dividing equation 11 by V_i . In 1D, the finite volume method reduces to a special case of finite differences:

$$\frac{\partial}{\partial t}(x_i) + \frac{1}{d\hat{\rho}}(\Gamma_{i+1/2} - \Gamma_{i-1/2}) = S_i, \quad (12)$$

In general, the fluxes in TORAX are decomposed as

$$\mathbf{\Gamma} = -D \frac{\partial x}{\partial \hat{\rho}} + V x \quad (13)$$

where D is a diffusion coefficient and V now denotes a convection coefficient, leading to:

$$\begin{aligned} \Gamma_{i+1/2} &= -D_{i+1/2} \frac{x_{i+1} - x_i}{d\hat{\rho}} + V_{i+1/2} x_{i+1/2} \\ \Gamma_{i-1/2} &= -D_{i-1/2} \frac{x_i - x_{i-1}}{d\hat{\rho}} + V_{i-1/2} x_{i-1/2} \end{aligned} \quad (14)$$

The diffusion and convection coefficients are thus calculated on the face grid. The value of x on the face grid is approximated by implementing a power-law scheme for Péclet weighting, which smoothly transitions between central differencing and upwinding, as follows:

$$\begin{aligned} x_{i+1/2} &= \alpha_{i+1/2} x_i + (1 - \alpha_{i+1/2}) x_{i+1} \\ x_{i-1/2} &= \alpha_{i-1/2} x_i + (1 - \alpha_{i-1/2}) x_{i-1} \end{aligned} \quad (15)$$

where the α weighting factor depends on the Péclet number, defined as:

$$Pe = \frac{Vd\hat{\rho}}{D} \quad (16)$$

where V is convection and D is diffusion. The power-law scheme is as follows:

$$\alpha = \begin{cases} \frac{Pe-1}{Pe} & \text{if } Pe > 10, \\ \frac{(Pe-1)+(1-Pe/10)^5}{Pe} & \text{if } 0 < Pe < 10, \\ \frac{(1+Pe/10)^5-1}{Pe} & \text{if } -10 < Pe < 0, \\ -\frac{1}{Pe} & \text{if } Pe < -10. \end{cases} \quad (17)$$

The Péclet number quantifies the relative strength of convection and diffusion. If the Péclet number is small and diffusion dominates, then the weighting scheme converges to central differencing. If the absolute value of the Péclet number is large, and convection dominates, then the scheme converges to upwinding. See figure 3 for an illustration of the α weighting factor dependence on the Péclet number in the power-law scheme.

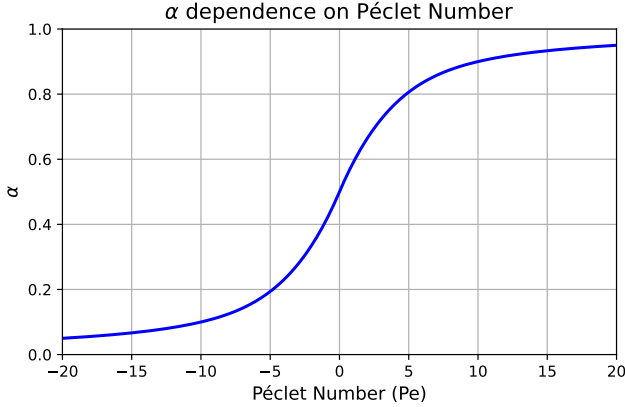


FIG. 3. α weighting factor dependence on the Péclet number in the power-law scheme, as according to equation 17.

Boundary conditions are taken into account by introducing ghost cells x_N and x_{-1} whose values are determined by assuming linear extrapolation through the edge cells and the face boundary conditions (for Dirichlet boundary conditions), or by directly satisfying the derivative (Neumann) boundary conditions. These ghost cells are substituted into equation 15 for the edge values.

Equations (14,15,17), when combined, define the elements of the discretization matrix and boundary condition vectors for the PDE diffusion term.

C. Solver implementation

TORAX time integration is based on the theta method, a widely used approach for time discretization of PDEs.

1. Theta method

The theta method is a weighted average between the explicit and implicit Euler methods. For a generic ODE of the form:

$$\frac{dx}{dt} = F(x, t) \quad (18)$$

where x is the state vector, the theta method approximates the solution at time $t + \Delta t$ as:

$$x_{t+\Delta t} - x_t = \Delta t [\theta F(x_{t+\Delta t}, t + \Delta t) + (1 - \theta) F(x_t, t)] \quad (19)$$

where θ is a user-selected weighting parameter in the range $[0, 1]$. Different values of θ correspond to well-known solution methods: explicit Euler ($\theta = 0$), Crank-Nicolson ($\theta = 0.5$), and implicit Euler ($\theta = 1$), which is unconditionally stable.

2. TORAX equation composition

Upon inspection of the TORAX governing equations 1-4, we generalize equation 19 and write the TORAX state evolution equation:

$$\begin{aligned} \tilde{\mathbf{T}}(x_{t+\Delta t}, u_{t+\Delta t}) \odot \mathbf{x}_{t+\Delta t} - \tilde{\mathbf{T}}(x_t, u_t) \odot \mathbf{x}_t = \\ \Delta t [\theta (\bar{\mathbf{C}}(x_{t+\Delta t}, u_{t+\Delta t}) \mathbf{x}_{t+\Delta t} + \mathbf{c}(x_{t+\Delta t}, u_{t+\Delta t})) \\ + (1 - \theta) (\bar{\mathbf{C}}(x_t, u_t) \mathbf{x}_t + \mathbf{c}(x_t, u_t))] \end{aligned} \quad (20)$$

Starting from an initial condition \mathbf{x}_0 , equation 20 solves for $\mathbf{x}_{t+\Delta t}$ at each timestep. \mathbf{x}_t is the evolving state vector at time t , including all variables being solved by the system, and is of length $\#N$, where $\#$ is the number of solved variables. For example, consider a simulation with a gridsize of 25 solving ion heat transport, electron heat transport, and current diffusion. Then $N = 25$, $\# = 3$, and \mathbf{x}_t is comprised of T_i , T_e , and ψ , each with its own set of N values, making a total vector length of 75.

\mathbf{u}_t corresponds to all known input parameters at time t . This includes boundary conditions, prescribed profiles (e.g. n_e in the example above), and input parameters such as heating powers or locations.

$\tilde{\mathbf{T}}$ is the transient term (following FiPy nomenclature), where \odot signifies element-wise multiplication. For example, for the T_e equation, $\tilde{\mathbf{T}} = \mathbf{n}_e$, which makes the system nonlinear if \mathbf{n}_e itself is an evolving variable.

$\bar{\mathbf{C}}(x_t, u_t)$ and $\bar{\mathbf{C}}(x_{t+\Delta t}, u_{t+\Delta t})$ are the discretization matrices, of size $\#N \times \#N$. In general, depending on the physics models used, $\bar{\mathbf{C}}$ depends on state variables \mathbf{x} , for example through state-variable dependencies of transport coefficients χ , D , V , plasma conductivity, and ion-electron heat exchange, making the system nonlinear due to the $x_{t+\Delta t}$ dependence. \mathbf{c} is a vector, containing source terms and boundary condition terms.

TORAX provides three solver options for solving equation 20, summarized next.

3. Linear solver

This solver addresses the nonlinearity of equation 20 with fixed-point iteration, also known as the predictor-corrector method. For K iterations (user-configurable), an approximation for $\mathbf{x}_{t+\Delta t}$ is obtained by solving the following equation iteratively with $k = 1, 2, \dots, K$:

$$\begin{aligned} & \tilde{\mathbf{T}}(x_{t+\Delta t}^{k-1}, u_{t+\Delta t}) \odot \mathbf{x}_{t+\Delta t}^k - \tilde{\mathbf{T}}(x_t, u_t) \odot \mathbf{x}_t = \\ & \Delta t [\theta (\bar{\mathbf{C}}(x_{t+\Delta t}^{k-1}, u_{t+\Delta t}) \mathbf{x}_{t+\Delta t}^k + \mathbf{c}(x_{t+\Delta t}^{k-1}, u_{t+\Delta t})) \\ & + (1 - \theta) (\bar{\mathbf{C}}(x_t, u_t) \mathbf{x}_t + \mathbf{c}(x_t, u_t))] \end{aligned} \quad (21)$$

and where $\mathbf{x}_{t+\Delta t}^0 = \mathbf{x}_t$.

By replacing $\mathbf{x}_{t+\Delta t}$ with $\mathbf{x}_{t+\Delta t}^{k-1}$ within the coefficients $\tilde{\mathbf{T}}$, $\bar{\mathbf{C}}$ and \mathbf{c} , these coefficients become known at every iteration step, making equation 21 a *linear* system of equations. $\mathbf{x}_{t+\Delta t}^k$ can then be solved using standard linear algebra methods implemented in JAX.

To further enhance the stability of the linear solver, particularly in the presence of stiff transport coefficients (e.g., when using the QLKNN turbulent transport model, see section III), the Pereverzev-Corrigan method [26] is implemented as an option. This method adds a large (user-configurable) artificial diffusion term to the transport equations, balanced by a large inward convection term such that zero extra transport is added at time t . These terms stabilize the solution, at the cost of accuracy over short transient phenomena, demanding care in the choice of Δt and the value of the artificial diffusion term.

4. Newton-Raphson solver

This solver solves the nonlinear equation 20, using a gradient-based iterative Newton-Raphson root-finding method for finding the value of $\mathbf{x}_{t+\Delta t}$ that renders the residual vector zero:

$$\mathbf{R}(\mathbf{x}_{t+\Delta t}, \mathbf{x}_t, \mathbf{u}_{t+\Delta t}, \mathbf{u}_t, \theta, \Delta t) = 0 \quad (22)$$

where \mathbf{R} is the LHS-RHS of equation 20.

Starting from an initial guess $\mathbf{x}_{t+\Delta t} = \mathbf{x}_{t+\Delta t}^0$, the Newton-Raphson method linearizes equation 22 about iteration $\mathbf{x}_{t+\Delta t}^k$ and solves the linear system for a step $\delta \mathbf{x}$:

$$\bar{\mathbf{J}}(\mathbf{x}_{t+\Delta t}^k) \delta \mathbf{x} = -\mathbf{R}(\mathbf{x}_{t+\Delta t}^k) \quad (23)$$

where $\bar{\mathbf{J}}$ is the Jacobian of \mathbf{R} with respect to $\mathbf{x}_{t+\Delta t}$. Crucially, JAX automatically calculates $\bar{\mathbf{J}}$ using auto-differentiation.

With $\delta \mathbf{x} = \mathbf{x}_{t+\Delta t}^{k+1} - \mathbf{x}_{t+\Delta t}^k$, $\mathbf{x}_{t+\Delta t}^{k+1}$ is solved using standard linear algebra methods implemented in JAX such as LU decomposition. This process iterates until the residual falls below a user-configurable tolerance ε , i.e.: $\|\mathbf{R}(\mathbf{x}_{t+\Delta t}^{k+1})\|_2 < \varepsilon$, where $\|\cdot\|_2$ is the vector two-norm.

Solver robustness is obtained with a combination of $\delta \mathbf{x}$ line search and Δt backtracking. $\delta \mathbf{x}$ line search reduces the step size within a given Newton iteration step, while Δt backtracking reduces the overall time step and restarts the entire Newton-Raphson solver for the present timestep, as follows:

- If a Newton step leads to an increasing residual, i.e. $\mathbf{R}(\mathbf{x}_{t+\Delta t}^{k+1}) > \mathbf{R}(\mathbf{x}_{t+\Delta t}^k)$, or if $\mathbf{x}_{t+\Delta t}^{k+1}$ is unphysical, e.g. negative temperature, then $\delta \mathbf{x}$ is reduced by a user-configurable factor, and the line-search checks are repeated. The total accumulative reduction factor in a Newton step is denoted τ .
- If during the line-search phase, τ becomes too low, as determined by a user-configurable variable, then the solve is abandoned and Δt backtracking is invoked. A new solve attempt is made at a reduced Δt , reduced by a user-configurable factor, which results in a less nonlinear system.

For the initial guess $\mathbf{x}_{t+\Delta t}^0$, two options are available. The user can start from \mathbf{x}_t , or use the result of the predictor-corrector linear solver as a warm-start.

5. Optimizer solver

An alternative nonlinear solver using the JAX-compatible jaxopt library [27] is also available. This method recasts the residual of equation 20 as a loss function, which is minimized using an iterative optimization algorithm. Similar to the Newton-Raphson solver, adaptive timestepping is implemented, where the timestep is reduced if the loss remains above a tolerance at exit. While offering flexibility with different optimization algorithms, this option is relatively untested for TORAX to date.

6. Timestep (Δt) calculation

TORAX provides two methods for calculating the timestep Δt , as follows.

- **Fixed Δt :** This method uses a user-configurable constant timestep throughout the simulation. If a nonlinear solver is employed, and adaptive timestepping is enabled, then in practice, some steps may have a lower Δt following backtracking.
- **Adaptive Δt :** This method adapts Δt based on the maximum heat conductivity $\chi_{\max} = \max(\chi_i, \chi_e)$. Δt is a multiple of a base timestep inspired by the explicit stability limit for parabolic PDEs:

$$\Delta t_{\text{base}} = \frac{(d\hat{p})^2}{2\chi_{\max}} \quad (24)$$

where $\Delta t = c_{\text{mult}}^{dt} \Delta t_{\text{base}}$. c_{mult}^{dt} is a user-configurable prefactor. In practice, c_{mult}^{dt} can be significantly larger than unity for implicit solution methods.

The adaptive timestep method protects against traversing through fast transients in the simulation, by enforcing $\Delta t \propto 1/\chi$.

III. IMPLEMENTED PHYSICS MODELS

TORAX provides a modular framework for incorporating various physics models. This section summarizes the initially implemented models for geometry, transport, sources, and neoclassical physics. Extended physics features are a focus of the short-term development roadmap. Specific plans are summarized following each subsection.

A. Magnetic geometry

TORAX presently supports two geometry models for determining the metric coefficients and flux surface averaged geometry variables required for the transport equations.

- **CHEASE:** This model utilizes equilibrium data from the CHEASE fixed boundary Grad-Shafranov equilibrium code [28]. Users provide a CHEASE output file, and TORAX extracts the relevant geometric quantities.
- **Circular:** For testing and demonstration purposes, TORAX includes a simplified circular geometry model. This model assumes a circular plasma cross-section and corrects for elongation to approximate the metric coefficients.

Using ψ and the magnetic geometry terms, the toroidal current density is calculated as follows:

$$j_{\text{tor}} = \frac{R_0^2}{8\pi^3\mu_0} \frac{\partial}{\partial V} \left(\frac{g_2 g_3 J}{\rho} \frac{\partial \psi}{\partial \rho} \right) \quad (25)$$

where V is the volume enclosed by a flux surface.

The safety factor q , a measure of magnetic field line pitch, is calculated as follows:

$$q = 2\pi B_0 \rho \left(\frac{\partial \psi}{\partial \rho} \right)^{-1} \quad (26)$$

With $q(\rho=0) = \frac{2B_0}{\mu_0 j_{\text{tot}}(\rho=0)R_0}$.

To enable simulations of tokamak scenarios with dynamic equilibria, the TORAX roadmap includes incorporating time-dependent geometry, e.g. by incorporating multiple equilibrium files and interpolating the geometry variables at the times required by the solver. Generalization to geometry data beyond CHEASE is also planned.

B. Plasma composition, initial and prescribed conditions

Presently, TORAX only accommodates a single main ion species, configured with its atomic mass number (A_i) and charge state (Z_i). The plasma effective charge, Z_{eff} , is assumed to be radially flat and is also user-configurable. A single impurity with charge state Z_{imp} is specified to accommodate $Z_{\text{eff}} > 1$. The main ion density dilution is then calculated as follows:

$$n_i = (Z_{\text{imp}} - Z_{\text{eff}})/(Z_{\text{imp}} - 1)n_e \quad (27)$$

Initial conditions for the evolving profiles T_i , T_e , n_e , and ψ are user-configurable. For $T_{i,e}$, both the initial core ($r=0$) value and the boundary condition at $\hat{\rho}=1$ are provided. Initial $T_{i,e}$ profiles are then a linear interpolation between these values.

For n_e , the user specifies a line-averaged density, in either absolute units or scaled to the Greenwald density $n_{\text{GW}} = \frac{I_p}{\pi a^2} [10^{20} \text{m}^{-3}]$, and the initial peaking factor. The initial n_e profile, including the edge boundary condition, is then a linear function scaled to match the desired line-averaged density and peaking.

If any of the $T_{i,e}$, n_e equations are set to be non-evolving (i.e., not evolved by the PDE stepper), then time-dependent prescribed profiles are user-configurable.

For the poloidal flux $\psi(\hat{\rho})$, the user specifies if the initial condition is based on a prescribed current profile, $j_{\text{tor}} = j_0(1 - \hat{\rho}^2)^\nu$ (with j_0 scaled to match I_p , and ν is user-configurable), or from the ψ provided in a CHEASE geometry file. The prescribed current profile option is always used for the circular geometry. The total current I_p can be user-configured or determined by the CHEASE geometry file. In the latter case, the ψ provided by CHEASE can still be used, but is scaled by the ratio of the desired I_p and the CHEASE I_p .

In the development roadmap, more flexible initial condition setting is planned, such as from a wider variety of formulas, from arbitrary arrays, or from arbitrary times from existing TORAX output files.

C. Transport models

Turbulent transport determines the values of the transport coefficients (χ_i , χ_e , D_e , V_e) in equations (1-3), and is a key ingredient in core transport simulations. Theory-based turbulent transport models provide the largest source of nonlinearity in the PDE system. TORAX currently offers three transport models:

- **Constant:** This simple model sets all transport coefficients to constant, user-configurable values. While not physically realistic, it can be useful for testing purposes.
- **CGM:** The critical gradient model (CGM) is a simple theory-based model, capturing the basic feature

of tokamak turbulent transport, critical temperature gradient transport thresholds [29]. The model is a simple way to introduce transport coefficient nonlinearity, and is mainly used for testing purposes.

$$\chi_i = \begin{cases} \chi_{GB} C \left(\frac{R}{L_{Ti}} - \frac{R}{L_{Ti,crit}} \right)^\alpha & \text{if } \frac{R}{L_{Ti}} \geq \frac{R}{L_{Ti,crit}} \\ \chi_{min} & \text{if } \frac{R}{L_{Ti}} < \frac{R}{L_{Ti,crit}} \end{cases} \quad (28)$$

with the GyroBohm scaling factor

$$\chi_{GB} = \frac{(A_i m_p)^{1/2} (T_i k_B)^{3/2}}{(e B_0)^2 R_{maj}} \quad (29)$$

and the Guo-Romanelli ion-temperature-gradient (ITG) mode critical gradient formula [30]

$$R/L_{Ti,crit} = \frac{4}{3} (1 + T_i/T_e) (1 + 2|\hat{s}|/q) \quad (30)$$

where χ_{min} is a user-configurable minimum allowed χ , $L_{Ti} \equiv -\frac{T_i}{\nabla T_i}$ is the ion temperature gradient length, A_i is the main ion atomic mass number, m_p the proton mass, e the electron charge, B_0 the magnetic field on axis, and R_{maj} the major radius. The stiffness factor C and the exponent α are user-configurable model parameters.

Regarding additional transport coefficient outputs, the electron heat conductivity, χ_e , and particle diffusivity, D_e , are scaled to χ_i using user-configurable model parameters. The particle convection velocity, V_e , is user-defined.

- **QLKNN:** This is a ML-surrogate model trained on a large dataset of the QuaLiKiz quasilinear gyrokinetic code [31, 32]. Specifically, TORAX presently employs the QLKNN-hyper-10D model (QLKNN10D) [14], which features a 10D input hypercube and separate NNs for ion-temperature-gradient (ITG), trapped-electron-mode (TEM), and electron-temperature-gradient (ETG) mode turbulent fluxes. The NNs take as input local plasma parameters, such as normalized gradients of temperature and density, temperature ratios, safety factor, magnetic shear, Z_{eff} , and normalized collisionality, and outputs turbulent fluxes for ion and electron heat and particle transport. The QLKNN model is significantly faster than direct gyrokinetic simulations, enabling fast and accurate simulation within its range of validity. The ability to seamlessly couple ML-surrogate models is a key TORAX feature. TORAX depends only on the open source weights and biases of the QLKNN model, and includes dedicated JAX inference code written with the Flax library [33].

For all transport models, optional spatial smoothing of the transport coefficients using a Gaussian convolution kernel is implemented, to improve solver convergence

rates, an issue which can arise with stiff transport coefficients such as from QLKNN. Furthermore, for all transport models, the user can set inner (towards the center) and/or outer (towards the edge) radial zones where the transport coefficients are prescribed to fixed values, overwriting the transport model outputs. This is useful, for example, to model the L-mode edge where QLKNN10D is not validated, or residual transport in the inner core area due to MHD (e.g. sawteeth) or slab or electromagnetic modes which QLKNN10D cannot capture [34].

The edge-transport-barrier, or pedestal, is a narrow region near the LCFS where turbulence is suppressed at sufficiently high input power, known as H-mode [35]. See figure 4 for an illustration. The pedestal effect can be mimicked in TORAX through an adaptive source term which sets a desired value (pedestal height) of T_e , T_i and n_e , at a user-configurable location (pedestal width).

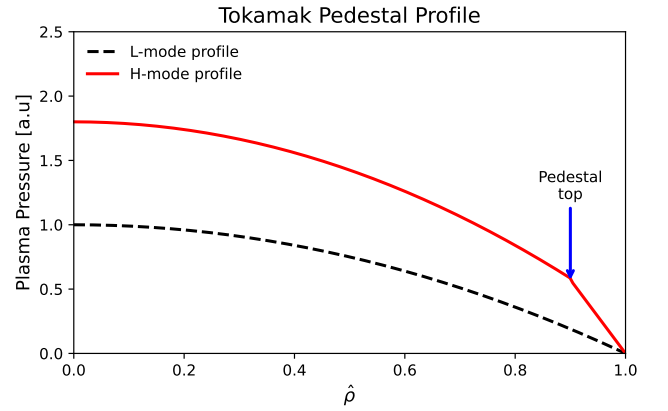


FIG. 4. Pedagogical illustration of a tokamak low confinement mode (L-mode) and high confinement mode (H-mode), with the latter characterised by an edge transport barrier with reduced transport, leading to locally high pressure gradients dubbed the “pedestal”.

In the TORAX roadmap, coupling to additional transport models is envisaged, including semi-empirical models such as Bohm/gyroBohm and H-mode confinement scaling law adaptive models, as well as more ML-surrogates of theory-based models, both for core turbulence and pedestal predictions. A more physically consistent approach for setting up the pedestal will be implemented by incorporating adaptive transport coefficients in the pedestal region, as opposed to an adaptive local source/sink term.

D. Neoclassical physics

TORAX employs the Sauter model [36] to calculate the bootstrap current density, j_{bs} , and the neoclassical conductivity, $\sigma_{||}$, used in the current diffusion equation (Eq. 4). The Sauter model is a widely-used analytical formulation that provides a relatively fast and differentiable approximation for these neoclassical quantities.

Future work can incorporate more recent neoclassical physics parameterizations [37], and also set neoclassical transport coefficients themselves. This can be of importance for ion heat transport in the inner core. When extending TORAX to include impurity transport, incorporating fast analytical neoclassical models for heavy impurity transport will be of great importance [38, 39].

E. Sources

The source terms in the governing equations 1-4 are comprised of a summation of individual source/sink terms. Each of these terms can be configured to be either:

- **Implicit:** Where needed in the theta-method, the source term is calculated based on the current guess for the state at $t + \Delta t$.
- **Explicit:** The source term is always calculated based on the state of the system at the beginning of the timestep, even if the solver $\theta > 0$. This makes the PDE system less nonlinear, avoids the incorporation of the source in the residual Jacobian if solving with Newton-Raphson, and leads to a single source calculation per timestep.

Explicit treatment is less accurate, but can be justified and computationally beneficial for sources with complex but slow-evolving physics. Furthermore, explicit source calculations do not need to be JAX-compatible, since explicit sources are an input into the PDE stepper, and do not require JIT compilation (see section IV). Conversely, implicit treatment can be important for accurately resolving the impact of fast-evolving source terms.

All sources can optionally be set to zero, prescribed with non-physics-based formulas (currently Gaussian or exponential) with user-configurable time-dependent parameters like amplitude, width, and location, or calculated with a dedicated physics-based model. Not all sources currently have a model implementation. However, the code modular structure facilitates easy coupling of additional source models in future work. Specifics of source models currently implemented in TORAX follow:

1. Ion-electron heat exchange

The collisional heat exchange power density is calculated as

$$Q_{ei} = \frac{1.5n_e(T_i - T_e)}{A_i m_p \tau_e}, \quad (31)$$

where A_i is the atomic mass number of the main ion species, m_p is the proton mass, and τ_e is the electron collision time, given by:

$$\tau_e = \frac{12\pi^{3/2}\epsilon_0^2 m_e^{1/2} (k_B T_e)^{3/2}}{n_e e^4 \ln \Lambda_{ei}}, \quad (32)$$

where ϵ_0 is the permittivity of free space, m_e is the electron mass, e is the elementary charge, and $\ln \Lambda_{ei}$ is the Coulomb logarithm for electron-ion collisions given by:

$$\ln \Lambda_{ei} = 15.2 - 0.5 \ln \left(\frac{n_e}{10^{20} \text{ m}^{-3}} \right) + \ln(T_e). \quad (33)$$

Q_{ei} is added to the electron heat sources, meaning that positive Q_{ei} with $T_i > T_e$ heats the electrons. Conversely, $-Q_{ei}$ is added to the ion heat sources.

2. Fusion power

TORAX optionally calculates the fusion power density assuming a 50-50 deuterium-tritium (D-T) fuel mixture using the Bosch-Hale parameterization [40] for the D-T fusion reactivity $\langle \sigma v \rangle$:

$$P_{fus} = E_{fus} \frac{1}{4} n_i^2 \langle \sigma v \rangle \quad (34)$$

where $E_{fus} = 17.6$ MeV is the energy released per fusion reaction, n_i is the ion density, and $\langle \sigma v \rangle$ is given by:

$$\langle \sigma v \rangle = C_1 \theta \sqrt{\frac{\xi}{m_r c^2 T_i^3}} \exp(-3\xi) \quad (35)$$

with

$$\theta = \frac{T_i}{1 - \frac{T_i(C_2 + T_i(C_4 + T_i C_6))}{1 + T_i(C_3 + T_i(C_5 + T_i C_7))}} \quad (36)$$

and

$$\xi = \left(\frac{B_G^2}{4\theta} \right)^{1/3} \quad (37)$$

where T_i is the ion temperature in keV, $m_r c^2$ is the reduced mass of the D-T pair. The values of $m_r c^2$, the Gamov constant B_G , and the constants C_1 through C_7 are provided in the Bosch-Hale paper.

TORAX partitions the fusion power between ions and electrons using the parameterized alpha particle slowing down model of Mikkelsen [41], which neglects the slowing down time itself.

3. Ohmic power

The Ohmic power density, P_{ohm} , arising from resistive dissipation of the plasma current, is calculated as:

$$P_{ohm} = \frac{j_{tor}}{2\pi R_{maj}} \frac{\partial \psi}{\partial t} \quad (38)$$

where j_{tor} is the toroidal current density, and R_{maj} is the major radius. The loop voltage $\frac{\partial \psi}{\partial t}$ is calculated according to (Eq. 4). P_{ohm} is then included as a source term in the electron heat transport equation.

4. Auxiliary Heating and Current Drive

While auxiliary heating such as neutral beam injection (NBI), ion cyclotron resonance heating (ICRH), etc, and their associated non-inductive current drives, can all be prescribed with formulas, presently no dedicated physics models are available within TORAX. Future work envisages incorporating more sophisticated physics-based models or ML-surrogate models, enhancing the fidelity of the simulation. For explicit sources, these can also come from external codes (not necessarily JAX compatible) coupled to TORAX in larger workflows.

Presently, a built-in non-physics-based Gaussian formulation of a generic ion and electron heat source is available in TORAX, with user configurable location, Gaussian width, and fractional heating of ions and electrons.

5. Particle Sources

Similar to auxiliary heating and current drive, particle sources can also be configured using either prescribed formulas. Presently, TORAX provides three built-in formula-based particle sources for the n_e equation:

- **Gas Puff:** An exponential function with configurable parameters models the ionization of neutral gas injected from the plasma edge.
- **Pellet Injection:** A Gaussian function approximates the deposition of particles from pellets injected into the plasma core. The time-dependent configuration parameter feature allows either a continuous approximation or discrete pellets to be modelled.
- **Neutral Beam Injection (NBI):** A Gaussian function models the ionization of neutral particles injected by a neutral beam.

Future work envisages coupling physics-based models and/or ML-surrogates.

6. Radiation

Currently, TORAX does not include dedicated models e.g. for cyclotron radiation, Bremsstrahlung, recombination, or line radiation are yet coupled to TORAX. This is left for future work.

IV. SOFTWARE DESIGN AND IMPLEMENTATION

The TORAX codebase is designed with modularity in mind, facilitating extension and maintenance.

A. Code Structure and Modularity

The main components of TORAX are organized into distinct modules:

- **Geometry:** Handles the representation and manipulation of tokamak geometry, including loading equilibrium data from external files and calculating geometric quantities required for the transport equations.
- **Transport Models:** Implements different transport models for calculating turbulent transport coefficients.
- **Sources:** Defines and manages the various sources and sinks driving the evolution of plasma profiles.
- **Steppers:** Provides numerical solvers for advancing the system of equations in time.
- **Time Step Calculators:** Determines the appropriate time step for each iteration of the solver.
- **State:** Defines data structures for representing the evolving plasma state, and output data structures representing physics and numerics outputs over the simulation time, such as transport coefficients, core profiles, sources, and stepper numerics.

The modules include constructor methods and class definitions for Python classes corresponding to simulation physical or numerical components such as **Geometry**, **TransportModel**, **Stepper**, **SourceProfiles**, which are passed throughout the code. Specific models or solvers are child classes of these objects. For example, a specific turbulent transport model is a child class of **TransportModel** with a concrete implementation of the call method, ensuring a unified API and shared methods such as the Gaussian smoothing kernel. This aids extensions to new models.

A **Sim** class, within the **sim** module, holds all components of the simulation and provides a method that runs the simulation macro time loop and associated glue code. This module is used by the standalone TORAX driver, or can be imported by external workflows to incorporate TORAX into wider simulation frameworks. See Algorithm 1 for an overview of the main simulation loop.

Algorithm 1 TORAX simulation procedure ('sim.run_simulation')

```

1:  $t \leftarrow t_{initial}$  ▷ Initialize simulation time
2: Initialize geometry, core_profiles, core_sources
3: while  $t < t_{final}$  do
4:   Calculate all explicit_source_profiles
5:    $\Delta t \leftarrow \text{TimeStepCalculator.next\_dt}(\dots)$  ▷ Calculate timestep
6:   Interpolate all prescribed quantities at  $t + \Delta t$ 
7:    $x_{t+\Delta t} \leftarrow \text{SimulationStepFn}(x_t, u_t, u_{t+\Delta t}, \Delta t, \dots)$  ▷ Advance state by one timestep
8:    $t \leftarrow t + \Delta t$  ▷ Update simulation time
9:   torax_outputs.append(sim_state) ▷ Append state to history for output
10: end while

```

In the abstraction above for the `SimulationStepFn` call, u contains the source runtime parameters, explicit sources (for u_t), geometry, prescribed profiles, and boundary conditions, whereas x is the evolving state variables. `SimulationStepFn` also contains the callables for the PDE stepper, the transport model, and associated runtime parameters.

B. Simulation I/O

1. Simulation input configuration

Users specify simulation parameters, such as boundary conditions, initial conditions, heating powers, and solver settings, by providing a Python nested dictionary. The config dictionary provides constructor arguments for constructing the various simulation objects, as well as their runtime parameters, and ultimately the TORAX `Sim` class. For testing, interactive experimentation, and custom workflows, it is also possible to circumvent the config dictionary and construct the `Sim` class manually.

TORAX supports time-dependent simulation inputs, enabling the modeling of dynamic tokamak scenarios. Users can define time series for various parameters on the config level, such as for plasma current, heating power, boundary conditions. TORAX implements either piecewise linear or stepwise interpolation schemes to handle these time-dependent inputs, as per user configuration per input.

2. Dynamic vs. Static Parameters

TORAX differentiates between dynamic and static parameters, affecting the JIT compilation behavior (see section IV C). Dynamic parameters can be modified between simulation runs without triggering recompilation, while static parameters define the fundamental structure of the simulation, and require recompilation if changed. These include the list of evolving variables, the grid size, the stepper functions, and transport models.

3. Simulation outputs

The outputs of the `run` method of the `Sim` class are dataclasses analogous to IMAS Interface Data Structures (IDSs), such as `core_profiles`, `core_sources`, and `core_transport`, containing a time history of all associated TORAX variables. In the native TORAX simulation driver, following the completion of the macro time loop, these variables are flattened, converted to an xarray Dataset object [42], and saved to disk as a netCDF file at a user-defined path. A basic visualization module is in place, to browse salient simulation outputs with a time-slider, including comparisons between different simulations simultaneously.

C. JAX JIT compilation and performance

TORAX uses JAX's just-in-time (JIT) compilation capabilities to achieve fast simulation runtimes. JIT compilation transforms Python functions into optimized XLA (Accelerated Linear Algebra) code, enabling efficient execution on CPUs, GPUs, and TPUs. The first call to a JIT-compiled function triggers a tracing and compilation process, which adds to the total simulation computation time. Subsequent calls do not recompile, and execute the optimized XLA code, bypassing the Python interpreter and achieving significant speedups. Compiled functions are cached in memory and can be used in subsequent TORAX simulation calls as long as the process is continued, even with new configs, as long as no static parameters are modified. For new processes, recompilation is necessary. Implementing the JAX persistent on-disk compilation cache in TORAX is currently under development. JIT compilation can be disabled in TORAX by an environment variable, aiding with interactive debugging.

Only the computationally intensive components of a TORAX simulation are compiled. This includes the source models, geometry calculations, neoclassical models, turbulent transport models, matrix manipulations, and the linear algebra solver. For the Newton-Raphson solver, the PDE residual function and its Jacobian are compiled. Crucially, the control flow logic of the iterative solver is not compiled, to avoid compilation inefficiencies, particularly with CPU. This design choice has advantages and disadvantages:

• Advantages:

- Significant compilation speedup (up to factor 10) on CPU.
- All control flow and glue code can be programmed in standard Python, which has less constraints. It makes it easier to handle input and output structures, plotting, logging, and coupling external non-JAX models (e.g. as explicit sources, or geometry) to the TORAX solver.

• Disadvantages:

- End-to-end differentiation of a TORAX simulation, e.g. taking the derivative of the final output state with respect to any input parameter, is not possible in a single step. However, the issue is mitigated by the PDE residual Jacobian being compiled and differentiable, making end-to-end differentiation possible via Forward Sensitivity Analysis. See section IV C 2.
- Developers need to be aware of the boundaries between elements in the codebase that need to be JAX compatible, and which are not, adding developer overhead.

1. Computational time for example simulation

We present a brief summary of simulation compilation and runtime performance, with different solver settings. All runs are based on the `iterhybrid_rampup.py` configuration file in the TORAX examples folder. This run evolves 80 seconds of ITER rampup (time-dependent I_p), with the QLKNN10D transport model, a gridsize of $N = 25$, and simulates ion and electron heat transport, particle transport, and current diffusion. A fixed $\Delta t = 2s$ is used. We compare performance when using the Newton-Raphson nonlinear solver, predictor-corrector with 1 corrector step, and predictor-corrector with 10 corrector steps. Simulations were performed on a single compute node equipped with a 48-core AMD EPYC 7B13 processor.

TABLE I. TORAX Simulation Performance Comparison

Solver	Compile [s]	Runtime [s]
Newton-Raphson	15.6	22
Predictor-Corrector (1 step)	4.5	6.5
Predictor-Corrector (10 steps)	4.6	8

Especially considering that compilation overhead is mitigated by in-memory and persistent caching, these runs demonstrate faster-than-realtime simulation capability. This is however not a general conclusion, since resolving transient phenomena on faster timescales, if needed, would slow down the simulation.

Figure 5 compares the three simulations. The 10-step predictor-corrector version converges to the proper nonlinear solution from Newton-Raphson, demonstrating that predictor-corrector can be a robust, fast solver option for use-cases where purely forward simulation is required, without residual and sensitivity information.

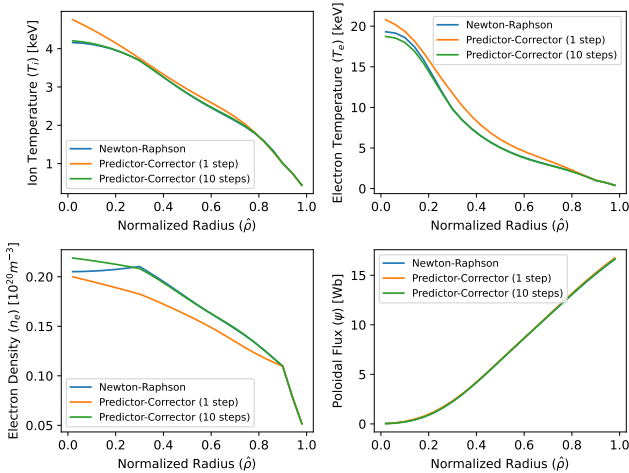


FIG. 5. Comparison of simulated T_i , T_e , n_e , and ψ profiles at $t = 10$ s for the Newton-Raphson, Predictor-Corrector (1 step), and Predictor-Corrector (10 steps) solvers, using the example `iterhybrid_rampup.py` configuration.

2. End-to-end differentiation

While the TORAX code design does not permit end-to-end direct differentiation, having a differentiable residual enables the application of the Forward Sensitivity Analysis Equation [23, 43], to obtain the sensitivity of the state to any input variable. This information can then be used for gradient-based optimization methods [8].

To illustrate this, let us parameterize part of the input vector u with a parameter p . For example, a parameterization of the ECRH power amplitude time trace. Following the application of the iterative solver, we obtain a state $x_{t+\Delta t}$ which is a solution of the residual:

$$\mathbf{R}(x_{t+\Delta t}, \mathbf{x}_t, \mathbf{u}_{t+\Delta t}, \mathbf{u}_t) = 0 \quad (39)$$

We now take the total derivative of the residual with respect to p , at timestep k .

$$0 = \frac{d\mathbf{R}_k}{dp} = \frac{\partial \mathbf{R}_k}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial p} + \frac{\partial \mathbf{R}_k}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial p} + \frac{\partial \mathbf{R}_k}{\partial \mathbf{u}_k} \frac{\partial \mathbf{u}_k}{\partial p} + \frac{\partial \mathbf{R}_k}{\partial \mathbf{u}_{k+1}} \frac{\partial \mathbf{u}_{k+1}}{\partial p} \quad (40)$$

We are interested in calculating $\frac{\partial \mathbf{x}_{k+1}}{\partial p}$, the sensitivity of the state at time $t + \Delta t$ to arbitrary input parameter. The key point, is that all the \mathbf{R} derivatives can be calculated by JAX in a generalized Jacobian, using auto-differentiation. All \mathbf{u} derivatives can also be calculated with JAX gradients.

Thus, starting from the initial condition $\frac{\partial \mathbf{x}_0}{\partial p}$, which is known, equation 40 is recursively solved until time $k + 1$. This method is not yet implemented in TORAX, but is on the short-term roadmap.

V. VERIFICATION AGAINST RAPTOR

To validate TORAX, we conducted benchmark simulations against the established RAPTOR code [8, 9], which has itself been validated against other transport solvers [14, 44].

The first comparison is for an ITER-like L-mode scenario, with constant plasma current of $I_p = 11.5$ MA, constant external power of $P_{tot} = 50$ MW equally split between ions and electrons, and using the default TORAX CHEASE ITER hybrid scenario equilibrium, as well as its ψ profile for the initial condition, scaled to match our requested I_p . Ion and electron heat transport, particle transport, and current diffusion were all modelled, for 10 seconds until stationary state for heat and particle transport, with $\Delta t = 0.05$ s. The constant transport model was used with $\chi_i/\chi_e = 2$, with inward convection included for the particle transport, and a broad particle source. Fusion power, bootstrap current, Ohmic power, and ion-electron heat exchange were all included

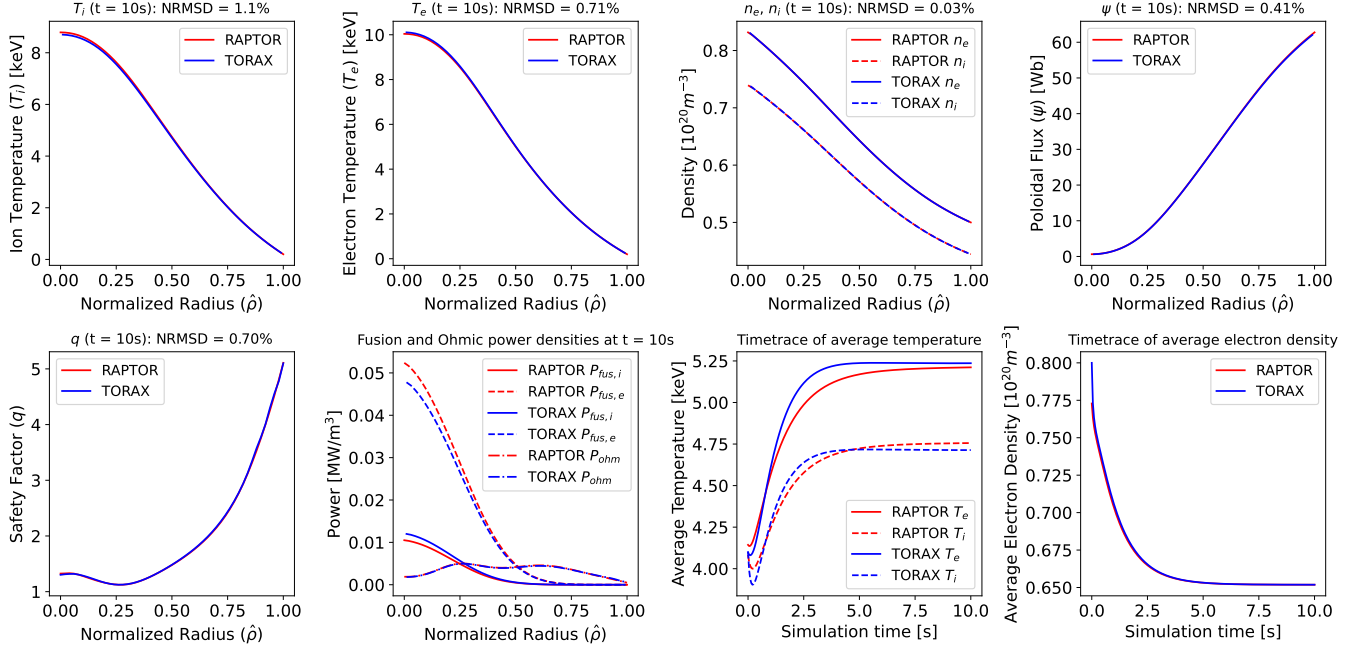


FIG. 6. Comparison of simulated profiles from TORAX and RAPTOR for an ITER L-mode scenario. The simulation includes ion and electron heat transport, particle transport, and current diffusion, using the constant transport coefficient model. The top row compares the ion and electron temperatures, electron density, and poloidal flux at the end of the simulation. The normalized root-mean-square deviation (NRMSD) between the TORAX and RAPTOR profiles, interpolated onto the same grid, is indicated in the title of each subplot, demonstrating excellent agreement between the two codes. The bottom row shows the safety factor (q -profile), and outputs from the Ohmic and fusion power models, at the end of the simulation. Minor differences in fusion power are attributed to the different models used for fractional power deposition to ions and electrons. The final panels in the bottom row show time traces of line-averaged electron and ion temperatures, and electron density. The temperature profiles have a maximum transient deviation of $\sim 2.5\%$.

in the simulation. The Newton-Raphson solver was used in TORAX. The TORAX configuration file is shown in table II.

Figure 6 shows a comparison between the simulated profiles from TORAX and RAPTOR. For quantitative assessment of the agreement, we calculated the normalized root-mean-square deviation (NRMSD) for each profile at $t = 10$ s, interpolated onto the same grid, using the formula:

$$\text{NRMSD} = \frac{\sqrt{\sum_{i=1}^N (y_{i,\text{TORAX}} - y_{i,\text{RAPTOR}})^2 / N}}{\sum_{i=1}^N (y_{i,\text{TORAX}}) / N} \times 100\% \quad (41)$$

where y represents the profile being compared (e.g., T_i , T_e , n_e), and N is the number of radial grid points.

The comparisons show excellent $\sim 1\%$ agreement at stationary state. Minor differences in fusion power at stationary state are due to the different models used for the fractional power deposition to ions and electrons. There is a maximum deviation of $\sim 2.5\%$ between the temperature profiles during the transient phase.

A further comparison was made to verify time-dependent inputs. A dynamic ITER L-mode scenario was simulated, using constant transport coefficients. Heat transport and current diffusion were modelled over

300 seconds of plasma evolution with 3 phases: a 100 second current rampup from 3 MA to 15 MA, a 100 second flattop phase, and a 100 second current rampdown back to 3 MA. 20 MW of pure electron heating was applied, ramping up and down during the current ramps. During the flattop phase, the power was modulated for 60 seconds, to trigger further dynamics. Plasma density was prescribed as flat, with a varying magnitude fixed to a Greenwald fraction of 0.5. The simulation is shown in figure 7. RAPTOR and TORAX track the dynamics closely, with a $\text{NRMSD} < 5\%$ for both T_e and T_i . The energy confinement timescale is on the order of seconds, evident by the fast (on discharge timescale) relaxation of the temperature profiles during the heating modulation. A longer timescale temperature increase is evident during flattop, which is the current diffusion timescale, leading to increased peaking of the Ohmic heating profile over time, and impacting the temperatures due to the constant transport coefficients.

VI. CONCLUSIONS AND FUTURE WORK

TORAX is a new open-source differentiable tokamak core transport simulator, offering significant advantages for fast and accurate tokamak scenario modeling, pulse

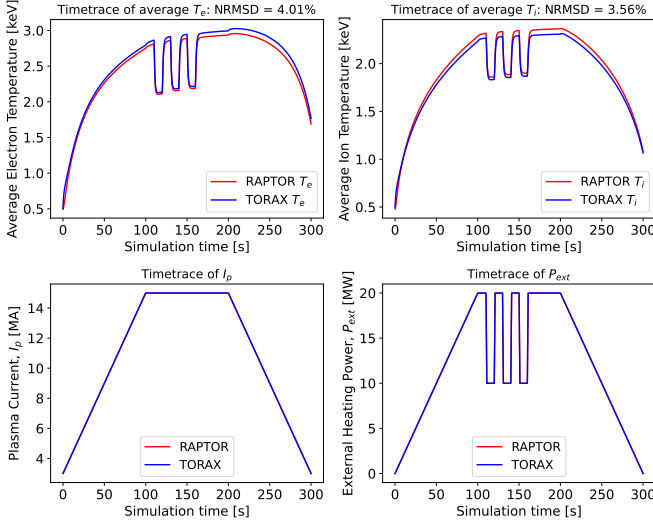


FIG. 7. Comparison of simulated profiles from TORAX and RAPTOR for an ITER L-mode scenario with varying current and input power. Constant transport coefficients are used, and the simulation includes ion and electron heat transport, and current diffusion, with a prescribed varying flat density profile set at a Greenwald fraction of 0.5. The bottom row illustrates the input plasma current and external electron heating power, identical for the two codes. Modulation in the heating power at current flat-top induces additional dynamics. The top row shows timetrace comparisons for ion and electron temperature. $NRMSE \approx 4\%$, with excellent agreement on the timescale dynamics over both energy confinement and current diffusion timescales.

design, and control. The choice of Python and the JAX library provides a flexible framework for implementing ML-surrogates of physics models, coupling within larger workflows, and fast simulation execution through compilation. TORAX is verified through comparison to the RAPTOR code, demonstrating good agreement in simulated plasma profiles, and any differences generally understood to arise from their different discretization and model implementations.

A key feature in TORAX is a differentiable PDE residual, enabling gradient-based nonlinear PDE solvers, and the use of the Forward Sensitivity Analysis Equation (see Eq. 40) to obtain the sensitivity of the state to arbitrary input parameters, facilitating gradient-based optimization workflows.

Upcoming technical development plans include:

- Time dependent geometry
- Increased flexibility for setting initial and prescribed conditions
- Implementation of forward sensitivity calculations w.r.t. control inputs and parameters
- Implementation of a persistent compilation cache
- Stationary-state solver
- Coupling to IMAS

The roadmap foresees the implementation of the following physics models:

- MHD models (e.g. sawteeth, neoclassical tearing modes)
- Radiation sinks (cyclotron, Bremsstrahlung, line radiation)
- Widened range of semi-empirical and theory-based ML-surrogates of turbulent transport models.
- Neoclassical transport and multi-ion transport, with a focus on heavy impurities
- Momentum transport

Contributions in line with the roadmap are welcome. In particular, TORAX is envisaged as a natural framework for coupling of various ML-surrogates of physics models. These could include surrogates for turbulent transport, neoclassical transport, heat and particle sources, line radiation, pedestal physics, core-edge integration, and MHD, among others.

Through its flexibility, speed, and ease of coupling ML-surrogates, the continued development of TORAX targets beyond-state-of-the-art applications for tokamak optimization, pulse-planning, and controller-design workflows.

VII. ACKNOWLEDGEMENTS

The TORAX authors thank Olivier Sauter and Antoine Merle for aiding with the provision of CHEASE equilibria, and Allen Wang for early contributions to the open source TORAX codebase.

Appendix A: Time dependent geometry impact on convective and source terms

For ion and electron heat transport, suppressing the species subscript and multiplying the equation by V' , we obtain

$$\frac{3}{2}V'^{-2/3}\left(\frac{\partial}{\partial t}-\frac{\dot{\Phi}_b}{2\Phi_b}\frac{\partial}{\partial\hat{\rho}}\right)\left[V'^{5/3}nT\right]=\frac{\partial}{\partial\hat{\rho}}\left[\chi n\frac{g_1}{V'}\frac{\partial T}{\partial\hat{\rho}}-g_0q^{\text{conv}}T\right]+V'Q \quad (\text{A1})$$

Moving the $\dot{\Phi}_b$ term to the RHS, we obtain:

$$\frac{3}{2}V'^{-2/3}\frac{\partial}{\partial t}\left(V'^{5/3}nT\right)=\frac{3}{4}V'^{-2/3}\frac{\dot{\Phi}_b}{\Phi_b}\frac{\partial}{\partial\hat{\rho}}\rho V'^{5/3}nT+\frac{\partial}{\partial\hat{\rho}}\left[\chi n\frac{g_1}{V'}\frac{\partial T}{\partial\hat{\rho}}-g_0q^{\text{conv}}T\right]+V'Q \quad (\text{A2})$$

Utilizing the identity $a\frac{\partial(bT)}{\partial\hat{\rho}}=\frac{\partial(abT)}{\partial\hat{\rho}}-bT\frac{\partial a}{\partial\hat{\rho}}$, and rearranging terms, we obtain:

$$\frac{3}{2}V'^{-2/3}\frac{\partial}{\partial t}\left(V'^{5/3}nT\right)=\frac{\partial}{\partial\hat{\rho}}\left[\chi n\frac{g_1}{V'}\frac{\partial T}{\partial\hat{\rho}}-\left(g_0q^{\text{conv}}-\frac{3}{4}\frac{\dot{\Phi}_b}{\Phi_b}\rho V'n\right)T\right]+V'Q+\frac{1}{2}V''\frac{\dot{\Phi}_b}{\Phi_b}\rho nT \quad (\text{A3})$$

Non-zero $\dot{\Phi}_b$ thus results in an effective convection term equal to $-\frac{3}{4}\frac{\dot{\Phi}_b}{\Phi_b}\rho V'n$, and an effective additional source term equal to $\frac{1}{2}V''\frac{\dot{\Phi}_b}{\Phi_b}\rho nT$, where V'' is the second derivative of the plasma volume with respect to $\hat{\rho}$.

For particle transport, using similar considerations, we obtain the following governing equation:

$$\frac{\partial}{\partial t}(n_e V')=\frac{\partial}{\partial\hat{\rho}}\left[D_en_e\frac{g_1}{V'}\frac{\partial n_e}{\partial\hat{\rho}}-\left(g_0V_e-\frac{\dot{\Phi}_b}{2\Phi_b}\rho V'\right)n_e\right]+V'S_n \quad (\text{A4})$$

where the $\dot{\Phi}_b$ terms leads to an additional effective convection term of $-\frac{\dot{\Phi}_b}{2\Phi_b}\rho V'$, and no additional effective source terms.

For current diffusion, we obtain:

$$\frac{16\pi^2\sigma_{||}\mu_0\dot{\Phi}_b^2}{F^2}\frac{\partial\psi}{\partial t}-\frac{\dot{\Phi}_b}{2\Phi_b}\frac{\partial\psi}{\partial\hat{\rho}}=\frac{\partial}{\partial\hat{\rho}}\left(\frac{g_2g_3}{\hat{\rho}}\frac{\partial\psi}{\partial\hat{\rho}}+8\pi^2\mu_0\Phi_b\dot{\Phi}_b\frac{\sigma_{||}\hat{\rho}^2}{F^2}\psi\right)-\frac{8\pi^2V'\mu_0\Phi_b}{F^2}\langle\mathbf{B}\cdot\mathbf{j}_{ni}\rangle-8\pi^2\mu_0\Phi_b\dot{\Phi}_b\psi\frac{\partial}{\partial\hat{\rho}}\left(\frac{\sigma_{||}\hat{\rho}^2}{F^2}\right) \quad (\text{A5})$$

where the $\dot{\Phi}_b$ terms leads to an additional effective convection term of $-8\pi^2\mu_0\Phi_b\dot{\Phi}_b\frac{\sigma_{||}\hat{\rho}^2}{F^2}$, and an additional effective source term of $-8\pi^2\mu_0\Phi_b\dot{\Phi}_b\psi\frac{\partial}{\partial\hat{\rho}}\left(\frac{\sigma_{||}\hat{\rho}^2}{F^2}\right)$.

-
- [1] Jozef Ongena, R Koch, R Wolf, and H Zohm. Magnetic-confinement fusion. *Nature Physics*, 12(5):398–410, 2016.
 - [2] Francesca Maria Poli. Integrated tokamak modeling: When physics informs engineering and research planning. *Physics of Plasmas*, 25(5):055602, 2018.
 - [3] A Fasoli, S Brunner, WA Cooper, JP Graves, P Ricci, O Sauter, and L Villard. Computational challenges in magnetic-confinement fusion physics. *Nature Physics*, 12(5):411–423, 2016.
 - [4] G. Cenacchi and A. Taroni. JETTO: A free boundary plasma transport code. Technical report, ENEA, Rome (Italy), 1988.
 - [5] Gregorij V Pereverzev, PN Yushmanov, A Yu Dnestrovskii, AR Polevoi, KN Tarasjan, and LE Zakharov. ASTRA—an automatic system for transport analysis in a tokamak. Technical report, Max-Planck-Institut fuer Plasmaphysik, Garching (Germany), 1991.
 - [6] Joshua Breslau, Marina Gorelenkova, Francesca Poli, Jai Sachdev, Alexei Pankin, Gopan Perumpilly, Xingqiu Yuan, Laszlo Glant, and USDOE Office of Science. Transp, 6 2018. URL <https://www.osti.gov/biblio/1489900>.
 - [7] Denis Kalupin, I Ivanova-Stanik, I Voitsekhovitch, J Ferreira, D Coster, LL Alves, Th Aniel, JF Artaud, V Ba-

- siuk, Joao PS Bizarro, et al. Numerical analysis of JET discharges with the European Transport Simulator. *Nuclear Fusion*, 53(12):123007, 2013.
- [8] F Felici and O Sauter. Non-linear model-based optimization of actuator trajectories for tokamak plasma profile control. *Plasma Physics and Controlled Fusion*, 54(2):025002, 2012.
- [9] F Felici, J Citrin, AA Teplukhina, J Redondo, C Bourdelle, F Imbeaux, O Sauter, JET Contributors, EUROfusion MST1 Team, et al. Real-time-capable prediction of temperature and density profiles in a tokamak using RAPTOR and a first-principle-based transport model. *Nuclear Fusion*, 58(9):096006, 2018.
- [10] Jean-François Artaud, Frédéric Imbeaux, J Garcia, G Giruzzi, T Aniel, V Basiuk, A Bécoulet, C Bourdelle, Y Buravand, J Decker, et al. Metis: a fast integrated tokamak modelling tool for scenario design. *Nuclear Fusion*, 58(10):105001, 2018.
- [11] F. Janky, E. Fable, M. Englberger, and W. Treutler. Validation of the fenix asdex upgrade flight simulator. *Fusion Engineering and Design*, 163:112126, 2021. ISSN 0920-3796. doi: <https://doi.org/10.1016/j.fusengdes.2020.112126>.
- [12] Shira M Morosohk, Andres Pajares, Tariq Rafiq, and Eugenio Schuster. Neural network model of the multi-mode anomalous transport module for accelerated transport simulations. *Nuclear Fusion*, 61(10):106040, 2021.
- [13] Orso Meneghini, Sterling P Smith, Philip B Snyder, Gary M Staebler, Jeffrey Candy, E Belli, L Lao, Mark Kostuk, T Luce, Teobaldo Luda, et al. Self-consistent core-pedestal transport simulations with neural network accelerated models. *Nuclear Fusion*, 57(8):086034, 2017.
- [14] Karel Lucas van de Plassche, Jonathan Citrin, Clarisse Bourdelle, Yann Camenen, Francis J Casson, Victor I Dagnelie, Federico Felici, Aaron Ho, Simon Van Mulders, and JET Contributors. Fast modeling of turbulent transport in fusion plasmas using neural networks. *Physics of Plasmas*, 27(2):022310, 2020.
- [15] Aaron Ho, J Citrin, CD Challis, C Bourdelle, FJ Casson, J Garcia, J Hobirk, A Kappatou, DL Keeling, DB King, et al. Predictive JET current ramp-up modelling using QuaLiKiz-neural-network. *Nuclear Fusion*, 63(6):066014, 2023.
- [16] MD Boyer, S Kaye, and K Erickson. Real-time capable modeling of neutral beam injection on NSTX-U using neural networks. *Nuclear Fusion*, 59(5):056008, 2019.
- [17] J Citrin, P Trochim, T Goerler, D Pfau, KL van de Plassche, and F Jenko. Fast transport simulations with higher-fidelity surrogate models for ITER. *Physics of Plasmas*, 30(6), 2023.
- [18] Semin Joung, Y-C Ghim, Jaewook Kim, Sehyun Kwak, Daeho Kwon, C Sung, D Kim, Hyun-Seok Kim, JG Bak, and SW Yoon. GS-DeepNet: mastering tokamak plasma equilibria with deep neural networks and the Grad-Shafranov equation. *Scientific Reports*, 13(1):15799, 2023.
- [19] P Rodriguez-Fernandez, NT Howard, and J Candy. Non-linear gyrokinetic predictions of SPARC burning plasma profiles enabled by surrogate modeling. *Nuclear Fusion*, 62(7):076036, 2022.
- [20] S Morosohk, B Leard, T Rafiq, and E Schuster. Machine learning-enhanced model-based scenario optimization for DIII-D. *Nuclear Fusion*, 64(5):056018, 2024.
- [21] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>.
- [22] F.L. Hinton and R.D. Hazeltine. Theory of plasma transport in toroidal confinement systems. *Rev. Mod. Phys.*, 48:239, 1976.
- [23] Federico Felici. *Real-time control of tokamak plasmas: from control of physics to physics-based control*. PhD thesis, EPFL, 2011.
- [24] Robert Eymard, Thierry Gallouët, and Raphaële Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.
- [25] Jonathan E. Guyer, Daniel Wheeler, and James A. Warren. FiPy: Partial differential equations with Python. *Computing in Science & Engineering*, 11(3):6–15, 2009. doi:10.1109/MCSE.2009.52. URL <http://www.ctcms.nist.gov/fipy>.
- [26] GV Pereverzev and G Corrigan. Stable numeric scheme for diffusion equation with a stiff transport. *Computer Physics Communications*, 179(8):579–585, 2008.
- [27] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.
- [28] Lutjens H., Bondeson A., and Sauter O. The CHEASE Code for Toroidal MHD Equilibria. *Comput. Phys. Commun.*, 97:219, 1996.
- [29] X Garbet, P Mantica, C Angioni, E Asp, Y Baranov, C Bourdelle, R Budny, F Crisanti, G Cordey, L Garzotti, et al. Physics of transport in tokamaks. *Plasma Physics and Controlled Fusion*, 46(12B):B557, 2004.
- [30] SC Guo and F Romanelli. The linear threshold of the ion-temperature-gradient-driven mode. *Physics of Fluids B: Plasma Physics*, 5(2):520–533, 1993.
- [31] C Bourdelle, J Citrin, B Baiocchi, A Casati, P Cottier, X Garbet, F Imbeaux, and JET Contributors. Core turbulent transport in tokamak plasmas: bridging theory and experiment with QuaLiKiz. *Plasma Physics and Controlled Fusion*, 58(1):014036, 2015.
- [32] J. Citrin, C. Bourdelle, F.J. Casson, C. Angioni, N. Bonanomi, Y. Camenen, X. Garbet, L. Garzotti, T. Görler, O. Gürcan, F. Koechl, F. Imbeaux, O. Linder, K.L. van de Plassche, P. Strand, and G. Szepesi JET contributors. Tractable flux-driven temperature, density, and rotation profile evolution with the quasilinear gyrokinetic transport model QuaLiKiz; <https://qualikiz.com>. *Plasma Physics and Controlled Fusion*, 59(12):124005, 2017.
- [33] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL <http://github.com/google/flax>.
- [34] Neeraj Kumar, Yann Camenen, Sadruddin Benkadda, Clarisse Bourdelle, Alberto Loarte, Alexei R Polevoi, Fabien Widmer, et al. Turbulent transport driven by kinetic ballooning modes in the inner core of JET hybrid H-modes. *Nuclear Fusion*, 61(3):036005, 2021.
- [35] F. Wagner, G. Becker, K. Behringer, D. Campbell, A. Eberhagen, W. Engelhardt, G. Fussmann, O. Gehre, J. Gernhardt, G. v. Gierke, G. Haas, M. Huang,

- F. Karger, M. Keilhacker, O. Klüber, M. Kornherr, K. Lackner, G. Lisitano, G. G. Lister, H. M. Mayer, D. Meisel, E. R. Müller, H. Murmann, H. Niedermeyer, W. Poschenrieder, H. Rapp, and H. Röhr. Regime of improved confinement and high beta in neutral-beam-heated divertor discharges of the asdex tokamak. *Phys. Rev. Lett.*, 49(19):1408–1412, Nov 1982. doi: 10.1103/PhysRevLett.49.1408.
- [36] O. Sauter, C. Angioni, and Y.R. Lin-Liu. Neoclassical conductivity and bootstrap current formulas for general axisymmetric equilibria and arbitrary collisionality regime. *Phys. Plasmas*, 6:2834, 1999.
- [37] A Redl, C Angioni, E Belli, O Sauter, ASDEX Upgrade Team, EUROfusion MST1 Team, et al. A new set of analytical formulae for the computation of the bootstrap current and the neoclassical conductivity in tokamaks. *Physics of Plasmas*, 28(2), 2021.
- [38] Clemente Angioni. Impurity transport in tokamak plasmas, theory, modelling and comparison with experiments. *Plasma Physics and Controlled Fusion*, 63(7): 073001, 2021.
- [39] Daniel Fajardo, Clemente Angioni, Francis J Casson, Anthony R Field, Patrick Maget, Pierre Manas, JET Contributors, ASDEX Upgrade Team, et al. Analytical model for the combined effects of rotation and collisionality on neoclassical impurity transport. *Plasma Physics and Controlled Fusion*, 65(3):035021, 2023.
- [40] H-S Bosch and GM Hale. Improved formulas for fusion cross-sections and thermal reactivities. *Nuclear fusion*, 32(4):611, 1992.
- [41] David R. Mikkelsen and Clifford E. Singer. Optimization of Steady-State Beam-Driven Tokamak Reactors. *Nuclear Technology - Fusion*, 4(2P1):237–252, 1983.
- [42] Stephan Hoyer and Hamman Joseph. xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1), April 2017. doi: 10.5334/jors.148.
- [43] Dan G Cacuci. Sensitivity theory for nonlinear systems. I. Nonlinear functional analysis approach. *Journal of Mathematical Physics*, 22(12):2794–2802, 1981.
- [44] Yong-Su Na, Florian Koechl, Alexei R Polevoi, Cheol-Sik Byun, DongHyeon Na, Jaemin Seo, Federico Felici, Atsushi Fukuyama, Jeronimo Garcia, Nobuhiko Hayashi, et al. On benchmarking of simulations of particle transport in iter. *Nuclear Fusion*, 59(7):076026, 2019.

TABLE II. TORAX configuration (Python dict) for the verification benchmark shown in figure 6.

```

CONFIG = {
  'runtime_params': {
    'plasma_composition': {
      'Ai': 2.5,
      'Zeff': 2.0,
      'Zimp': 10,
    },
    'profile_conditions': {
      'Ip': 11.5,
      'Ti_bound_left': 8,
      'Ti_bound_right': 0.2,
      'Te_bound_left': 8,
      'Te_bound_right': 0.2,
      'ne_bound_right': 0.5,
      'nbar_is_fGW': False,
      'nbar': 0.8,
      'npeak': 1.0,
      'set_pedestal': False,
    },
    'numerics': {
      't_final': 10.0,
      'exact_t_final': True,
      'fixed_dt': 0.05,
      'ion_heat_eq': True,
      'el_heat_eq': True,
      'current_eq': True,
      'dens_eq': True,
    },
  },
  'geometry': {
    'nr': 50,
    'geometry_type': 'chease',
    'geometry_file': 'ITER_hybrid_citrin_equil_cheasedata.mat2cols',
    'Ip_from_parameters': True,
    'Rmaj': 6.2,
    'Rmin': 2.0,
    'B0': 5.3,
  },
  'sources': {
    'j_bootstrap': {},
    'nbi_particle_source': {
      'S_nbi_tot': 3e+21,
      'nbi_deposition_location': 0.2,
      'nbi_particle_width': 0.25,
    },
    'generic_ion_el_heat_source': {
      'rsource': 0.11,
      'w': 0.2,
      'Ptot': 50e6,
      'el_heat_fraction': 0.5,
    },
    'ohmic_heat_source': {},
    'fusion_heat_source': {},
    'qei_source': {},
  },
  'transport': {
    'transport_model': 'constant',
    'constant_params': {
      'chii_const': 2.0,
      'chie_const': 1.0,
      'De_const': 1.0,
      'Ve_const': -0.15,
    },
  },
  'stepper': {
    'stepper_type': 'newton_raphson',
    'predictor_corrector': True,
    'corrector_steps': 5,
  },
  'time_step_calculator': {
    'calculator_type': 'fixed',
  },
}

```