

TORAX - A Fast and Differentiable Tokamak Transport Simulator in JAX

**Jonathan Citrin, Sebastian Bodenstein,
Anushan Fernando, Ian Goodfellow, Philippe Hamel,
Tamara Norman, Akhil Raju, Craig Donner, Federico Felici,
Andrea Huber, David Pfau, Brendan Tracey**

Google DeepMind

Devon Battaglia, Anna Teplukhina, Josiah Wai

Commonwealth Fusion Systems



Photo by Khyati Trehan for Google DeepMind on Unsplash

Science Portfolio at Google DeepMind

- Apply AI/ML to tackle challenging problems in science with significant impact
- Track record of open science for societal benefit (e.g. AlphaFold)



Article | [Open access](#) | Published: 17 January 2024

Solving olympiad geometry without human demonstrations

[Trieu H. Trinh](#), [Yuhuai Wu](#), [Quoc V. Le](#), [He He](#) & [Thang Luong](#)

Nature 625, 476–482 (2024) | [Cite this article](#)



Pushing the frontiers of density functionals by solving the fractional electron problem

[James Kirkpatrick](#), [Brendan McMorrow](#), [David H. P. Turban](#), [Alexander L. Gaunt](#), [L.] and [Arón J. Cohen](#) +12 authors [Authors info & Affiliations](#)

SCIENCE • 9 Dec 2021 • Vol 374, Issue 6573 • pp. 1385-1389 • DOI:10.1126/science.abb6511

Article | [Open access](#) | Published: 29 November 2023

Scaling deep learning for materials discovery

[Amil Merchant](#), [Simon Batzner](#), [Samuel S. Schoenholz](#), [Muratahan Aykol](#), [Gwoon Cheon](#) & [Ekin Dogus Cubuk](#)

Nature 624, 80–85 (2023) | [Cite this article](#)



Article | [Open access](#) | Published: 16 February 2022

Magnetic control of tokamak plasmas through deep reinforcement learning

[Jonas Degraeve](#), [Federico Felici](#), [Jonas Buchli](#), [Michael Neuner](#), [Brendan Tracey](#), [Francesco Carpanese](#), [Timo Ewalds](#), [Roland Hafner](#), [Abbas Abdolmaleki](#), [Diego de las Casas](#), [Craig Donner](#), [Leslie Fritz](#), [Cristian Galperti](#), [Andrea Huber](#), [James Keeling](#), [Maria Tsimpoukelli](#), [Jackie Kay](#), [Antoine Merle](#), [Jean-Marc Moret](#), [Seb Noury](#), [Federico Pesamosca](#), [David Pfau](#), [Olivier Sauter](#), [Cristian Sommariva](#), ... [Martin Riedmiller](#) + Show authors




Nature 602, 414–419 (2022) | [Cite this article](#)

Fusion at GDM

- Building on success of deep reinforcement learning for pulse + magnetic controller design
- Next: multiphysics, fast and accurate simulation environment for multi-objective optimization

Article | [Open access](#) | [Published: 16 February 2022](#)

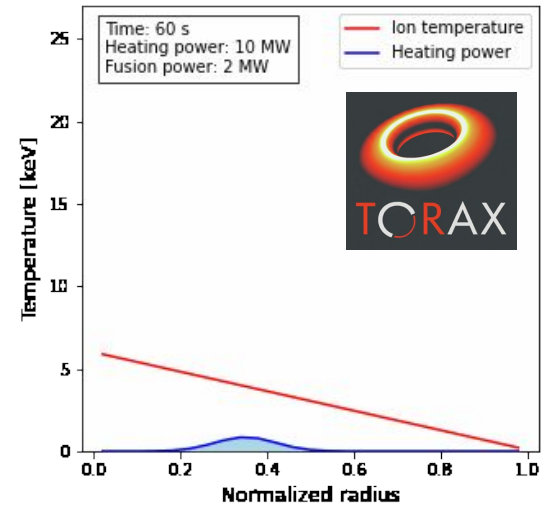
Magnetic control of tokamak plasmas through deep reinforcement learning

[Jonas Degraeve](#), [Federico Felici](#) , [Jonas Buchli](#) , [Michael Neunert](#), [Brendan Tracey](#) , [Francesco Carpanese](#), [Timo Ewalds](#), [Roland Hafner](#), [Abbas Abdolmaleki](#), [Diego de las Casas](#), [Craig Donner](#), [Leslie Fritz](#), [Cristian Galperti](#), [Andrea Huber](#), [James Keeling](#), [Maria Tsimpoukelli](#), [Jackie Kay](#), [Antoine Merle](#), [Jean-Marc Moret](#), [Seb Noury](#), [Federico Pesamosca](#), [David Pfau](#), [Olivier Sauter](#), [Cristian Sommariva](#), ... [Martin Riedmiller](#)  Show authors

[Nature](#) 602, 414–419 (2022) | [Cite this article](#)



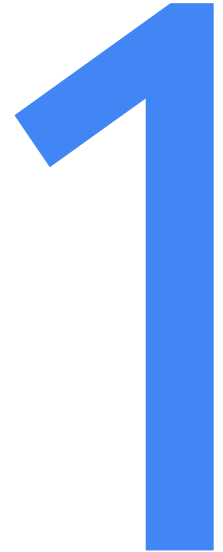
TORAX is our new **differentiable tokamak core transport simulator** built in Python using JAX, solving for core temperatures, density, and current diffusion



Outline

TORAX and JAX overview	01
Detailed model description	02
Benchmarks vs RAPTOR	03
Outlook	04

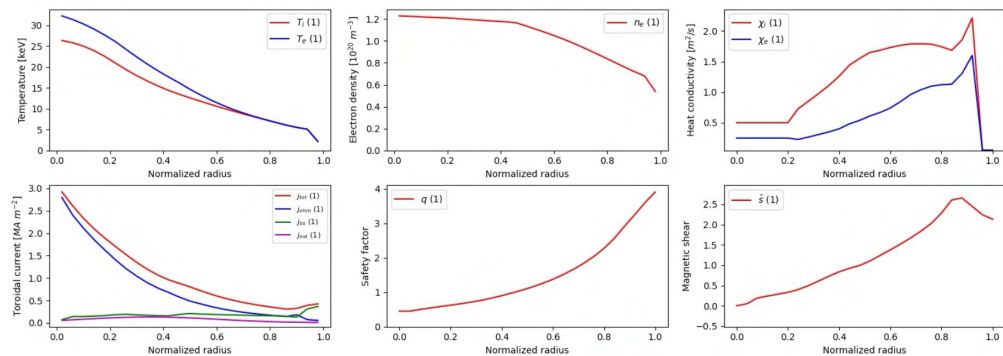
TORAX and JAX overview







TORAX is a differentiable tokamak core transport simulator aimed for fast and accurate forward modelling, pulse-design, trajectory optimization, and controller design workflows

- Core transport code
 - Heat and particle transport, current diffusion
- Python
 - Core solver is with JAX
 - Just-in-time compiled (fast)
 - Autodifferentiable
- Numerics
 - Finite-volume discretization
 - Implicit timestep PDE solver
 - Predictor-corrector method
 - Nonlinear solver with Newton-Raphson
- Physics
 - Analytical models + ML-surrogates for speed
 - Also supports standard integrated modelling mode with flexible model coupling and JAX optionally disabled

ITER baseline scenario with QLKNN10D.
Parameters based on Mantica PPCF 2020



TORAX motivated by requirements for pulse simulation, planning, and controller design tasks

-  Fast and accurate forward modelling
-  Differentiable for accurate nonlinear PDE solvers, gradient-driven optimization and parameter identification
-  Easy incorporation of physics model ML-surrogates; higher fidelity simulation without sacrificing speed
-  Modularity for coupling within various workflows and to additional physics models.

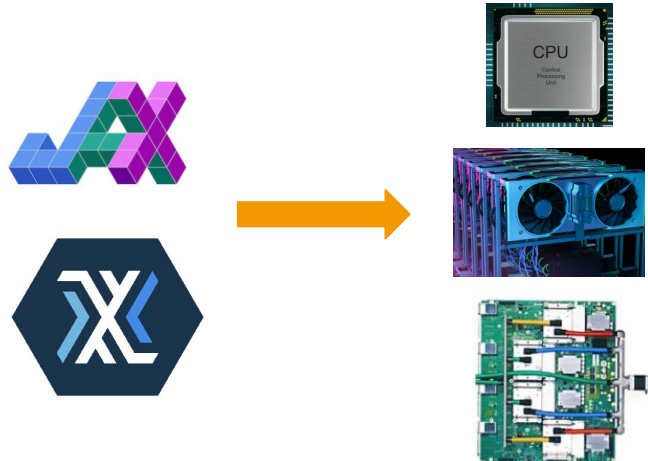
Building on ideas developed over many years in the fusion community for tokamak scenario modeling, pulse planning, and control [RAPTOR, JINTRAC, ASTRA, TRANSP, etc]

TORAX a natural evolution of the approach pioneered by RAPTOR, with advantages

- JAX auto-differentiation instead of manually derived Jacobians as in RAPTOR [Felici PPCF 2012]
 - Easy to extend simulator with any complex analytical model, e.g. ML surrogates
 - Easy to extend sensitivity analysis with new parameter inputs
- ML-surrogate coupling facilitated by JAX's inherent support for NN development and inference
 - Combine "medium/high fidelity" accuracy with "low fidelity" speed
- Python-as-a-feature
 - Freely available and widespread
 - Development velocity
 - Facilitates coupling within various simulation environments
- Running on accelerators (GPU) enables large-scale batch simulations

JAX enables fast compiled and differentiable simulation with NumPy-like API

- JAX, originally developed by Google, is "NumPy on the CPU, GPU, TPU" (<https://jax.readthedocs.io/>)
 - Originally developed for AI/ML. Increasing applied for scientific computing
- Uses an updated version of Autograd to automatically differentiate NumPy-like code
 - Automatic transformation of functions to their *analytic* derivatives
- Uses TensorFlow's XLA (Accelerated Linear Algebra) JIT (just in time) compiler for speed
 - Same code can run seamlessly on CPU or accelerators



Simple example of JAX function transformation



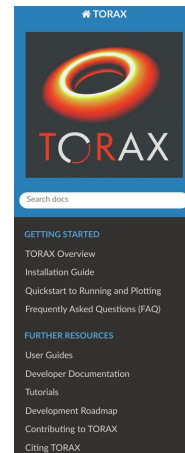
```
def sum_logistic(x):  
    return jnp.sum(1.0 / (1.0 + jnp.exp(-x)))  
  
x_small = jnp.arange(3.)  
derivative_fn = grad(sum_logistic)  
print(derivative_fn(x_small))
```

```
[0.25      0.19661194  0.10499357]
```

- Functions can be transformed into just-in-time (jit) compiled versions
 - e.g. `jitted_sum_logistic = jax.jit(sum_logistic)`
- Compile time overheads; for TORAX $O(\text{compile-time}) \approx O(\text{run-time})$
 - Mitigated by compilation caching on memory or persistent (disk)
- In TORAX, bottleneck functions are JAX, e.g. PDE residual calculation + its Jacobian.
 - Glue code, control-flow, pre+post processing is standard Python.
Eases development + coupling to wider frameworks.

TORAX open-source with permissive Apache 2.0 license

- Open source launch in June 2024
 - <https://github.com/google-deepmind/torax>
- Technical report + online documentation
 - torax.readthedocs.io
 - <https://arxiv.org/abs/2406.06718>



🏠 / TORAX: Tokamak transport simulation in JAX View page source

TORAX: Tokamak transport simulation in JAX

TORAX is a differentiable tokamak core transport simulator aimed for fast and accurate forward modelling, pulse-design, trajectory optimization, and controller design workflows. TORAX is written in Python using the JAX library.

Flexible

Python facilitates coupling within various workflows and to additional physics models. Easy to install and JAX can seamlessly execute on multiple backends including CPU and GPU.

Fast and auto-differentiable

JAX provides just-in-time compilation for fast runtimes. JAX auto-differentiability enables gradient-based nonlinear PDE solvers and simulation sensitivity analysis while avoiding the need to manually derive Jacobians.

ML-surrogates

ML-surrogate coupling for fast and accurate simulation is greatly facilitated by JAX's inherent support for neural network development and inference.

Getting Started

User Guides

Developer Docs

Next

© Copyright 2024, The TORAX Authors.



arXiv > physics > arXiv:2406.06718

Physics > Plasma Physics

[Submitted on 10 Jun 2024 (v1), last revised 12 Jun 2024 (this version, v2)]

TORAX: A Fast and Differentiable Tokamak Transport Simulator in JAX

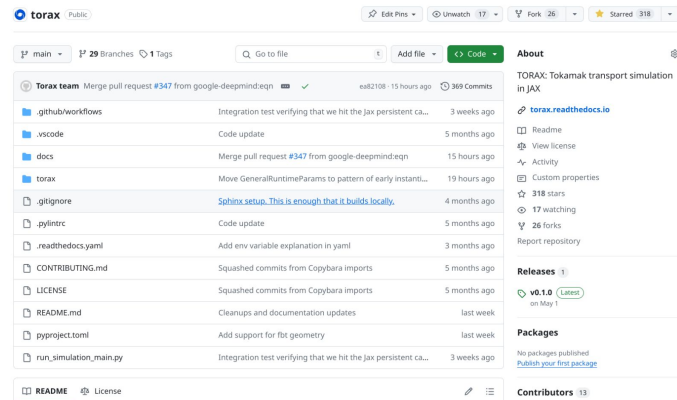
Jonathan Citrin, Ian Goodfellow, Akhil Raju, Jeremy Chen, Jonas Degraeve, Craig Donner, Federico Felici, Philippe Hamel, Andrea Huber, Dmitry Nikulin, David Pfau, Brendan Tracey, Martin Riedmiller, Pushmeet Kohli

We present TORAX, 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 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 solver 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 versatile tool for accelerating research in tokamak scenario modeling, pulse design, and control.

Comments: 16 pages, 7 figures

Subjects: Plasma Physics (physics.plasm-ph)

Cite as: arXiv:2406.06718 [physics.plasm-ph]
(or arXiv:2406.06718v2 [physics.plasm-ph] for this version)
<https://doi.org/10.48550/arXiv.2406.06718>



Detailed model description

- Governing equations
- Implementation of numerics
- Physics models

Includes "roadmaps" for planned work ahead.

Not exhaustive and does not include possible additional contributions from wider community, e.g. coupling new ML-surrogates of physics models



Governing equations: set of 1D flux-surface-averaged nonlinear transport PDEs

Ion and electron heat equations

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

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

Electron density equation

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

Current diffusion equation

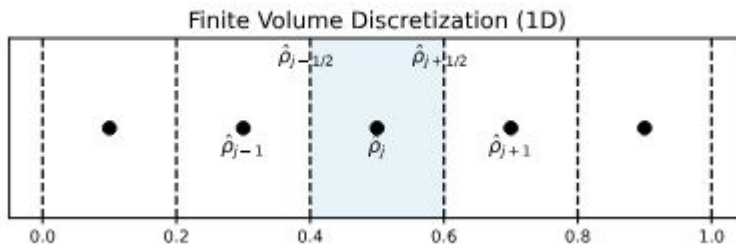
$$\frac{16\pi^2 \sigma_{||} \mu_0 \hat{\rho} \Phi_b^2}{F^2} \left(\frac{\partial \psi}{\partial t} - \frac{\hat{\rho} \dot{\Phi}_b}{2\Phi_b} \frac{\partial \psi}{\partial \hat{\rho}} \right) = \frac{\partial}{\partial \hat{\rho}} \left(\frac{g_2 g_3}{\hat{\rho}} \frac{\partial \psi}{\partial \hat{\rho}}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

- Polymorphic data input (xarray, numpy, Python primitives) for setting initial conditions, and optional "prescribed" profiles chosen not to be evolved by PDE system
- Zero-gradient boundary conditions on axis.
 T_i, T_e, n_e : Dirichlet boundary conditions at edge
 ψ : Neumann boundary condition at edge (l_p)
- Z_{eff} radial profile with single impurity.

Roadmap

- V_{loop} boundary condition for ψ : sets $\psi_{\text{LCFS}}(t+\Delta t)$ (WIP by UKAEA)
- Extension to multiple ions/impurities evolution
- Momentum transport

Spatial discretization: finite-volume-method with bespoke JAX fvm package



- TORAX JAX fvm package inspired by FiPy¹ and uses similar API
- Evolving profiles are CellVariable class with various helper methods, different options for boundary conditions, etc.
- For particle convection, power-law α -weighting scheme based on Péclet number
- Constructs nonlinear/linear systems of equations for solvers based on solution method

¹<https://www.ctcms.nist.gov/fipy/>

Temporal discretization and composition of system numerically solved

Theta method, first-order in time.

$\theta=1$, fully-implicit (default)

$$\mathbf{x}_{t+\Delta t} - \mathbf{x}_t = \Delta t [\theta F(\mathbf{x}_{t+\Delta t}, t+\Delta t) + (1-\theta)F(\mathbf{x}_t, t)]$$

Governing equations more generally decomposed as follows

$$\frac{3}{2}V'^{-5/3} \left(\frac{\partial}{\partial t} - \frac{\dot{\Phi}_b}{2\Phi_b} \frac{\partial}{\partial \hat{\rho}} \right) [V'^{5/3} n_i T_i] = \frac{1}{V'} \frac{\partial}{\partial \hat{\rho}} \left[\chi_i n_i \frac{g_1}{V'} \frac{\partial T_i}{\partial \hat{\rho}} - g_0 q_i^{\text{conv}} T_i \right] + Q_i \longrightarrow \tilde{\mathbf{T}}(\mathbf{x}_{t+\Delta t}, \mathbf{u}_{t+\Delta t}) \odot \mathbf{x}_{t+\Delta t} - \tilde{\mathbf{T}}(\mathbf{x}_t, \mathbf{u}_t) \odot \mathbf{x}_t = \Delta t [\theta (\bar{\mathbf{C}}(\mathbf{x}_{t+\Delta t}, \mathbf{u}_{t+\Delta t}) \mathbf{x}_{t+\Delta t} + \mathbf{c}(\mathbf{x}_{t+\Delta t}, \mathbf{u}_{t+\Delta t})) + (1-\theta) (\bar{\mathbf{C}}(\mathbf{x}_t, \mathbf{u}_t) \mathbf{x}_t + \mathbf{c}(\mathbf{x}_t, \mathbf{u}_t))]$$

- \mathbf{x} is state vector, subset of $\{T_i, T_e, n_e, \psi\}$, at time t or $t+\Delta t$
- $\tilde{\mathbf{T}}$ is the "transient term", e.g. $V'^{5/3} n_i$ in ion heat equation
- $\bar{\mathbf{C}}$ is the discretization matrix, including (possibly state-dependent) physics quantities like transport coefficients, geometry, etc.
- \mathbf{c} is vector with source terms + boundary condition terms
- \mathbf{u} corresponds to all "known" or quantities at time t and $t+\Delta t$, e.g. boundary conditions, heating amplitude trajectories, prescribed profiles, etc.

Solver methods

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

- Linear solver: replace $\mathbf{x}_{t+\Delta t}$ with \mathbf{x}_t in $\tilde{\mathbf{T}}, \bar{\mathbf{C}}, \mathbf{c}$
- Predictor corrector: fixed point iteration for k (user-defined) steps on $\mathbf{x}_{t+\Delta t}$ in $\tilde{\mathbf{T}}, \bar{\mathbf{C}}, \mathbf{c}$, starting from initial guess \mathbf{x}_t

- Newton-Raphson nonlinear solver: iterative root finding for residual

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

Where \mathbf{R} is the LHS-RHS of (1)

- Cast as optimization problem for $\mathbf{x}_{t+\Delta t}$, i.e. minimize loss = R^2 , using JAX optimization libraries, e.g. jaxopt (relatively unexplored)

Newton-Raphson illustration: simple example with heat diffusion

$$\frac{\partial T_k}{\partial t} = \frac{\partial}{\partial r} \left(\chi(T_{k+1}) \frac{\partial T_{k+1}}{\partial r} \right) + S(T_{k+1})$$

Fully implicit simple nonlinear diffusion equation

$$\frac{\vec{T}_{k+1} - \vec{T}_k}{\Delta t} - \bar{A}(\chi(T_{k+1})) \vec{T}_{k+1} - \vec{S}(T_{k+1}) \equiv \vec{R}(T_{k+1}, T_k) = 0$$

Discretize and define nonlinear system of equations to be solved (residual)

$$\vec{R}(T_{old}, T_k) + \bar{J}|_{(\vec{T}_{old})} (\vec{T}_{new} - \vec{T}_{old}) = 0$$

Newton-Raphson: starting from initial guess of T_{old} (e.g. T_k), iteratively solve linear system for T_{new} , until $R(T_{new}, T_k)$ within tolerance

All the physics goes into the residual function, and then JAX magic

```
jacobian = jax.jacfwd(residual)
```

- TORAX has simple linesearch to ensure good Newton steps (physical T_{new}), as well as Δt backtracking if no convergence

Presently implemented physics models/couplings: geometry

- Ad-hoc analytical "circular" geometry with elongation
- Numerical equilibria from Grad-Shafranov solvers
 - Flux surface averaged geometric quantities
 - Supports CHEASE, MEQ (FBT), EQDSK (Adam Kit @ VTT)
- Various initialization options
 - ψ directly from geometry file: can rescale to desired I_p
 - ψ consistent with user-provided current profile
- Assumes input sequence of pre-calculated equilibria.

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

$$g_1 = \langle (\nabla V)^2 \rangle$$

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

$$g_3 = \left\langle \frac{1}{R^2} \right\rangle$$

Roadmap

- Short-term: Initialize ψ based on q-profile
- IMAS equilibrium IDS input - need to wait for IMAS open-sourcing
- Medium term: Support inter-timestep coupling with GS solver for self-consistency with transport solution
 - Ideally with ML-surrogate to maintain speed

Presently implemented physics models/couplings: transport

- Turbulent Transport
 - Ad-hoc: constant coefficients, ITG Critical-Gradient-Model based on Guo Romanelli 1993
 - BohmGyroBohm semi-empirical model (Theo Brown @ UKAEA)
 - QuaLiKiz-neural-network (QLKNN10D hypercube version [van de Plassche 2020])
 - QuaLiKiz ("standard" integrated modelling mode with JAX-compilation disabled)
- Supports inner/outer transport patches + Gaussian smoothing kernel (important for convergence)
- Prescribed pedestal height and width maintained by adaptive source term
- Neoclassical resistivity and bootstrap current

Roadmap

- QLKNN11D under development based on existing dataset [<https://zenodo.org/records/8011148>]
- H-factor scaling models
- Neoclassical transport
 - e.g. FACIT for heavy impurities once impurity transport implemented
- Adaptive transport coefficients for pedestal as opposed to adaptive source
- Sawtooth and NTM models
- Couple more ML-surrogates from community, e.g. TGLF, higher-fidelity GK surrogates

Presently implemented physics models/couplings: sources

- DT fusion power (Bosch-Hale + Mikkelsen fraction power model)
- Ohmic power
- Equipartition (ion-electron heat exchange)
- Bremsstrahlung
- Lin-Liu model for ECCD with prescribed Gaussian heat deposition profile (Theo Brown @ UKAEA)
- Supports generic formulas (e.g. gaussian, exponential) for ad-hoc heat, particle, current sources
- TORIC-NN for SPARC configuration space (Greg Wallace @ MIT) - imminent

Non-JAX source models can be included as "explicit_sources", e.g. calculated outside step function using \mathbf{x}_t state and passed as an argument into the PDE solver, with the rest of the PDE still solved with JAX.

Roadmap

- Cyclotron radiation
- Line radiation and charge-state-equilibria models
- Couple additional ML-surrogates from community

Comparison with other integrated modelling suites

	e.g. JETTO/ASTRA	RAPTOR	TORAX
Coding language	Fortran	MATLAB	Python w/JAX
Linear solver + predictor-corrector	Yes	No	Yes
Newton-Raphson nonlinear solver	No	Yes	Yes
Discretization	FVM	FEM	FVM
Time-stepping	Adaptive	Deterministic	Adaptive or deterministic
Differentiable	No	Yes: manual	Yes: automatic
ML-surrogate coupling	Bespoke Fortran interface	Mex-files from Fortran	Python w/JAX
Range of physics models	Broad, high-fidelity, no restriction (apart from speed)	Parameterized: formulas, ML-surrogates	<p>Any: analytical models + ML-surrogates can be in JAX kernel. Non-JAX models can be injected into kernel as explicit terms</p> <p>High-fidelity models can still be coupled into solver kernel: JAX compilation can be disabled.</p> <p>Allows evaluation of ML-surrogates against ground truth within the same framework</p>

Runtimes and solver comparison

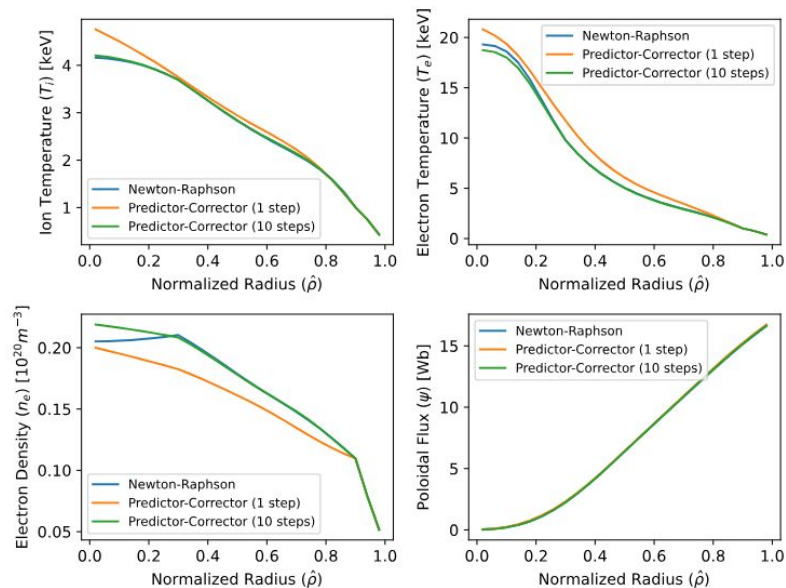


FIG. 5. Comparison of simulated T_i , T_e , n_e , and ψ profiles at $t = 10$ s for the Newton-Raphson, Predictor-Corrector (1 step), and Predictor-Corrector (10 steps) solvers, using the example `iter-hybrid_rampup.py` configuration.

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

Runtime and solver comparisons for a 80s ITER hybrid scenario rampup using the QLKNN transport model.

- Faster than realtime for this config
- Multi-step predictor-corrector can be attractive solver mode when gradient information from Jacobian not needed
- Still some low hanging fruit for runtime performance optimization

TORAX config design facilitates use and coupling in various workflows

- Config imported from Python module
 - Allows pre-processing of input data in standard Python
 - Hierarchical nested CONFIG dict passed into TORAX, which then constructs all simulation runtime_params
- Flexibility in prescribed input data
 - Can have any time resolution. Interpolated to correct time at every TORAX (non-deterministic) timestep
 - Data structures can be Python primitives, numpy, xarray
 - IMAS coupling facilitated by existing open-source IMAS <-> xarray libraries. Future work.

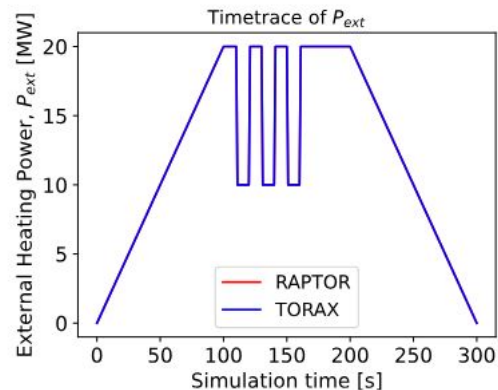
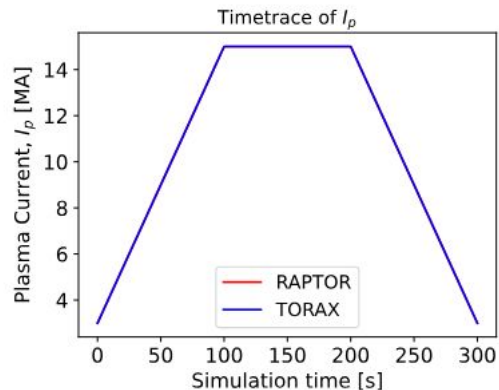
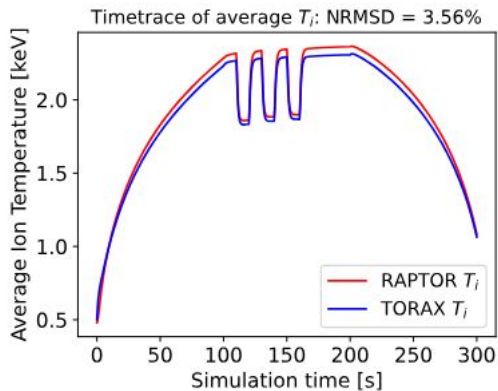
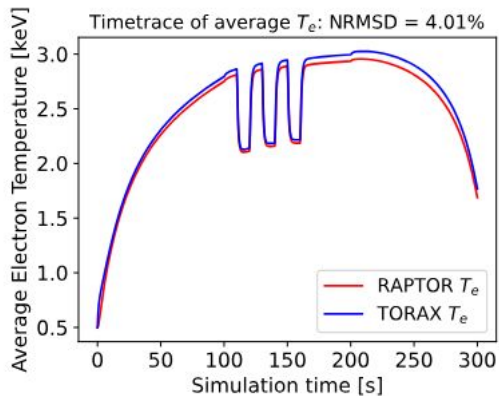
```
# numpy versions of prescribed quantities
Ip = (LY_times, LY_Ip)
ne = (tgrid, rhogauss, ne_time_dependent)
Te_initial = (rhogauss, Te_initial_values)
Ti_initial = (rhogauss, Ti_initial_values)
Tped = (tgrid, Tped)
ne_bound_right = (tgrid, ne_time_dependent[:, -1])

CONFIG = {
    'runtime_params': {
        'plasma_composition': {
            'Ai': 2.0,
            'Zeff': Zeff,
            'Zimp': Zimp,
        },
        'profile_conditions': {
            'Ip': Ip,
            'initial_psi_from_j': True,
            'initial_j_is_total_current': True,
            'nu': 1,
            'Ti': Ti_initial, # Initial condition only
            'Ti_bound_right': T_rhoedge,
            'Te': Te_initial, # Initial condition only
            'Te_bound_right': T_rhoedge,
            'ne_is_fGW': False,
            'normalize_to_nbar': False,
            'ne': ne,
            'ne_bound_right': ne_bound_right,
            'set_pedestal': ({0: False, t_LH: True, t_HL: False}, 'STEP'),
            'Tped': Tped,
            'Tped': Tped,
            'Ped_top': 0.95,
        },
        'numerics': {
            't_final': 11.0,
            'exact_t_final': True,
            'fixed_dt': 0.2,
            'ion_heat_eq': True,
            'el_heat_eq': True,
            'current_eq': True,
            'dens_eq': False,
        }
    }
}
```


Benchmarks vs RAPTOR

3

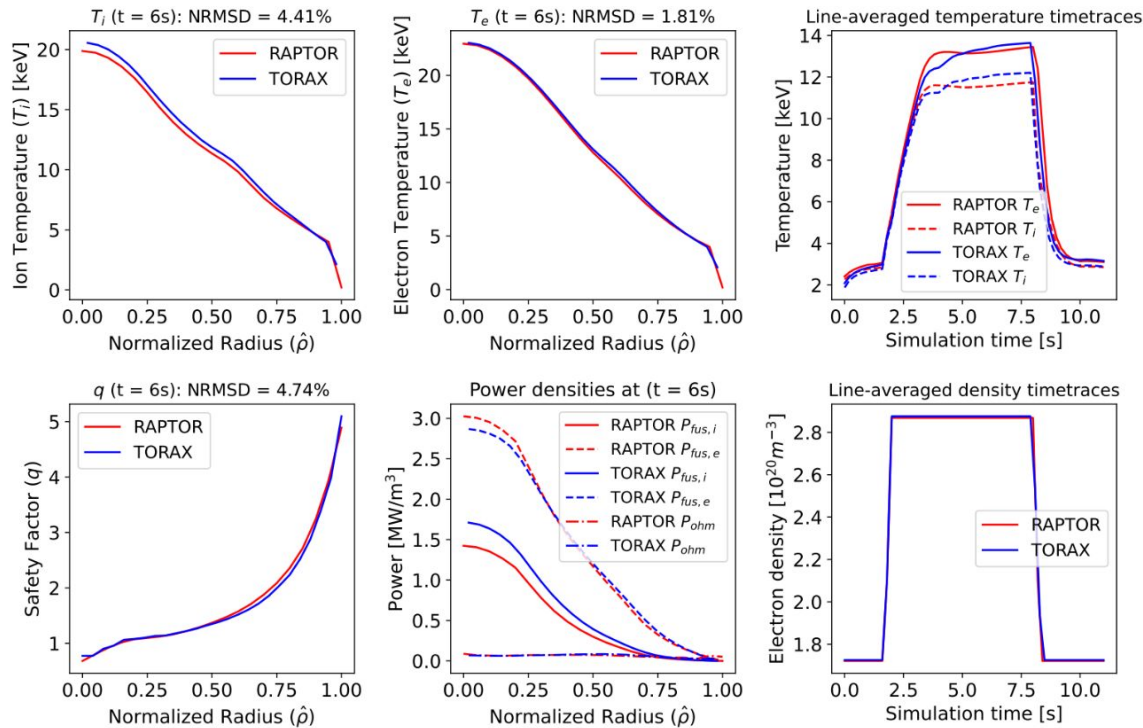
Verification: TORAX vs RAPTOR agreement for ITER-like cases



Modeling settings:

- ITER hybrid-scenario inspired params
- Nonlinear Newton-Raphson solver
- L-mode
- Heat transport + current diffusion
- CHEASE equilibrium
- 20MW ECRH at flat top: modulated
- Constant transport coefficients
- No sawteeth

Verification: TORAX vs RAPTOR agreement on SPARC H-mode scenario



Modeling settings:

- SPARC full-pulse H-mode
- Nonlinear Newton-Raphson solver
- Heat transport + current diffusion
- Sequence of FBT equilibria
- 11MW ICRH at flat top
- QLKNN10D* ML-surrogate transport
- No sawteeth

$\Delta t = 0.2\text{s}$: RAPTOR walltime: ~70s, TORAX walltime: ~14s

Outlook

4

Roadmap: towards higher physics fidelity and pulse planning applications

Short term developments

- Finalize RAPTOR parity
 - Sawteeth
 - End-to-end differentiable simulation with Forward Sensitivity Analysis
- Coupling to the CFS MOSAIC Pulse Planner (see Wai, Battaglia, APS 2024)
- Enable use for STEP (UKAEA)
 - Benchmarks with JETTO

Priorities for improved physics

- Multi-ion + impurity transport
 - Line radiation
 - Neoclassical transport
- Improved ML-surrogates
 - Turbulence
 - Pedestal
 - Sources
 - Edge
- Core-edge integration

Validation against data

- Open source datasets for integrated modelling validation highly valuable
- Motivated to engage in community effort for this goal

Calculating the Jacobian of the PDE system residual, with respect to any arbitrary simulation parameter, is greatly facilitated by autodiff

State x_{k+1} solves residual at time k , e.g. with iterative Newton method.
 u is control input, parameterized by p (e.g. ECCD power waveform)

$$\tilde{f}_k \equiv \tilde{f}(x_{k+1}, x_k, u_k) = 0 \quad \forall k$$

Forward Sensitivity Analysis method

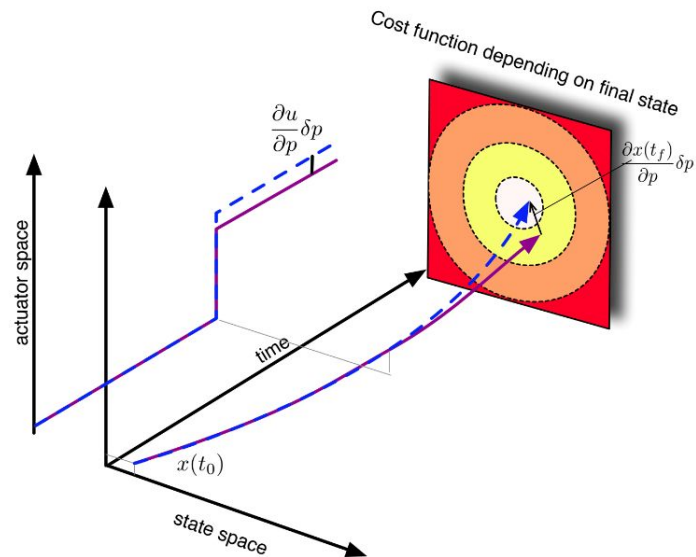
$$0 = \frac{d\tilde{f}_k}{dp} = \frac{\partial \tilde{f}_k}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial p} + \frac{\partial \tilde{f}_k}{\partial x_k} \frac{\partial x_k}{\partial p} + \frac{\partial \tilde{f}_k}{\partial u_k} \frac{\partial u_k}{\partial p} + \frac{\partial \tilde{f}_k}{\partial p}$$

We want $\frac{\partial x_{k+1}}{\partial p}$, how the solution changes with respect to a control input modification.

Linear system above is recursively solved starting from initial condition $\frac{\partial x_0}{\partial p}$

All f derivatives known and come from autodiff!

Key tool for sensitivity analysis, data-driven parameter identification, trajectory optimization methods



Summary

- TORAX, a new Python-based core transport code
 - With JAX: Fast for many-query applications, and autodifferentiable
 - Targeted for easy coupling to range of ML-surrogates
 - Verified against RAPTOR
- Open-source for wider community impact
 - Glad to support use and applications
 - Glad to support integration of new physics models and ML-surrogates
 - Open source data for validation; keen to engage community on this effort
- Can couple to broader fusion simulation frameworks
 - Speed supports various applications in forward and inverse modelling
- Excited to see TORAX in action!



<https://arxiv.org/abs/2406.06718>
<https://github.com/google-deepmind/torax>
<https://torax.readthedocs.io>