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

Fusion at Google DeepMind

Demonstrated deep reinforcement learning for pulse trajectory + magnetic controller design

Next: more physics for multi-objective optimization and controller design tasks

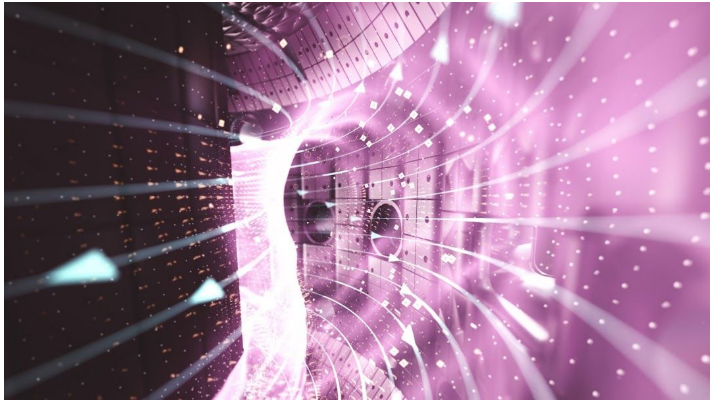
TORAX in this context: fast-and-accurate simulation environment for internal plasma dynamics

Article | Open access | Published: 16 February 2022

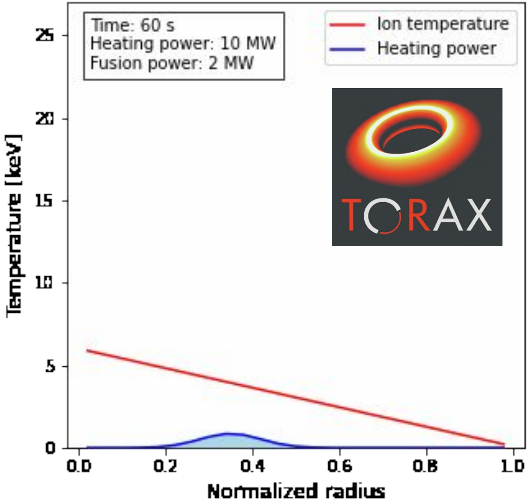
Magnetic control of tokamak plasmas through deep reinforcement learning

Jonas Degrave, 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

Motivations	01
TORAX overview	02
Benchmarks	03
Outlook	04

Motivations

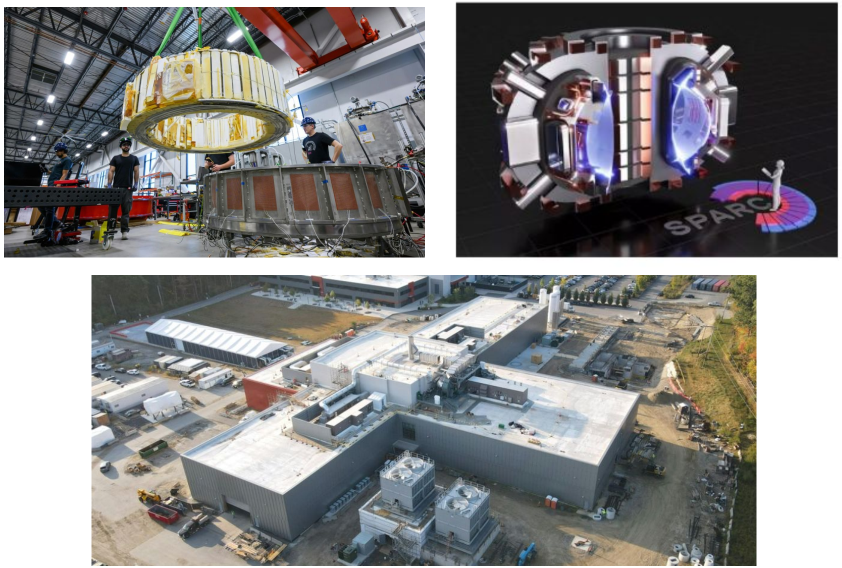
1

Next generation of tokamak experiment aiming for net fusion gain

ITER: Cadarache, France, standard 5T field

Acknowledgement to ITER Organization

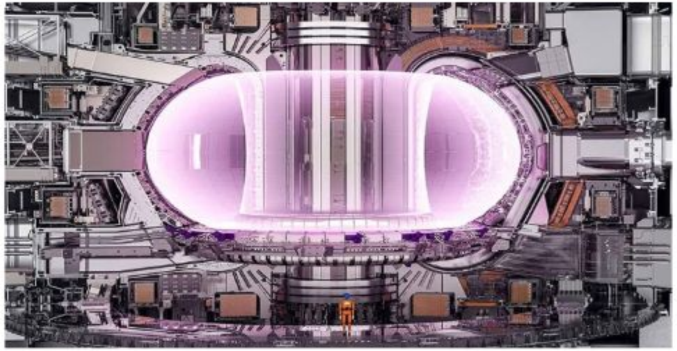
SPARC, Commonwealth Fusion Systems (CFS), HFS. ~12T

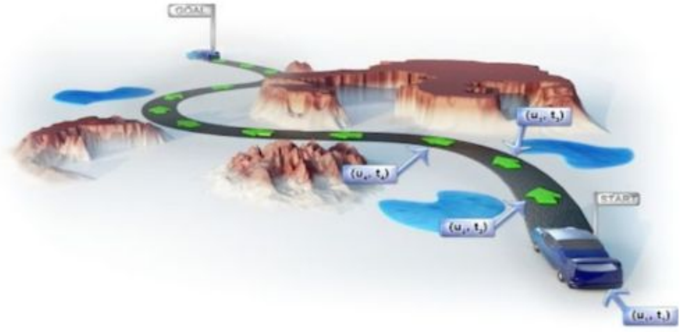


Acknowledgement to Commonwealth Fusion Systems

Key issue: present-day physics models too slow for routine simulations used for experiment prediction and interpretation

Leap from present-day experiments to reactors requires leap in simulation capabilities





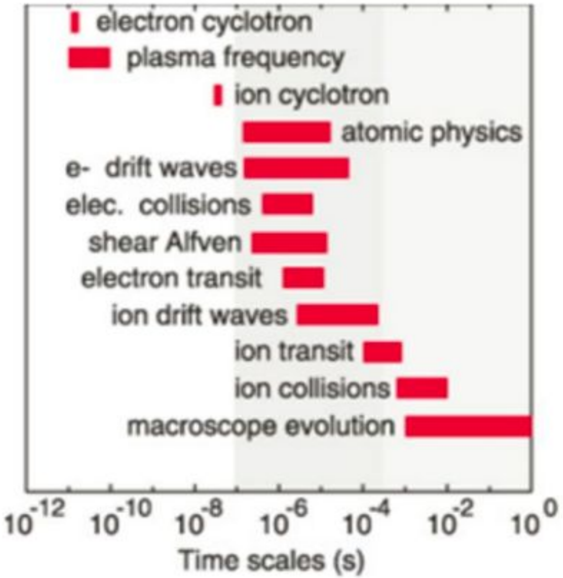
Reduce costs and risks with simulations for:

- Experimental preparation including inter-shot
- Performance (constrained) optimization
- Model based controller design
- Reactor design

In next generation devices, fast and accurate simulation a prerequisite for pulse design

Integrated modelling inherently multiscale and multiphysics

Multiple orders of magnitude in spatiotemporal scales between relevant physics processes



Magnetic equilibrium

Plasma collisions

Magnetohydrodynamics

Heating and fuelling

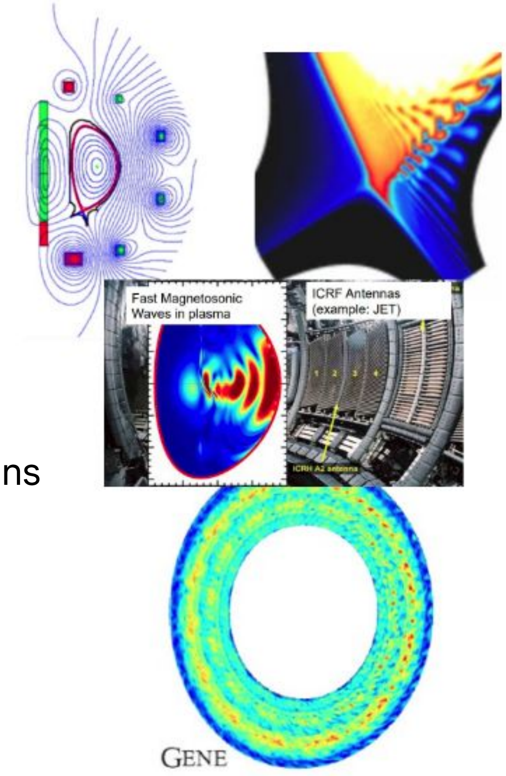
Plasma turbulence

Atomic and molecular interactions

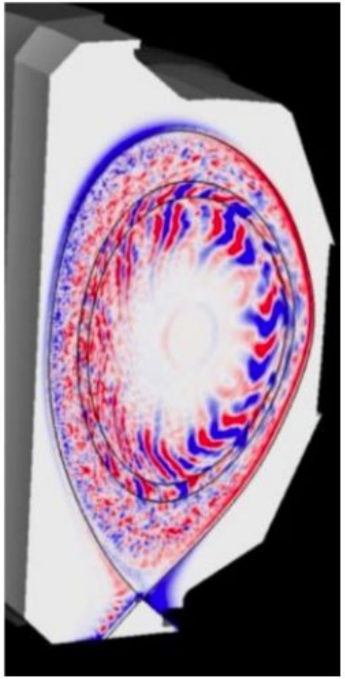
Plasma material interaction

…

…



Modelling fidelity and tractability hierarchy. Pragmatic modelling demands model reduction

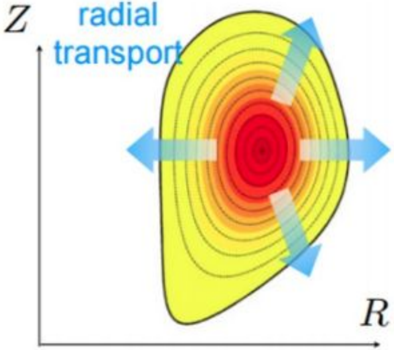


Dominiski Phys. Plasmas 2018

Simulation Class	Method	Fidelity level	Compute burden	Advantages	Challenges
Direct numerical simulation	Coupling high fidelity models	High	Massively parallel and expensive (exascale)	Ultimate ground truth (numerically)	Not pragmatic for most use cases
Standard integrated modelling	Plasma transport PDEs + coupled reduced models	Variable (depends on quality of reduced models)	Hours to weeks on single compute node	Suitable for experimental interpretation and extrapolation Community workhorse. Lots of experience and models available	Legacy burden. Tractability and accuracy are conflicting constraints
Fast integrated modelling	Plasma transport PDEs + coupled surrogate models	Variable (depends on quality of surrogates)	Faster than realtime to minutes	Suitable for optimization and controller design applications. Surrogates can learn higher fidelity models than “standard”	Need to develop collection of learned surrogates + appropriate framework

Integrated modelling simulation environment:
Separate plasma regions in core (~1D) and edge (~2D).
Different computational challenges that must be integrated

Plasma core:
1D coupled PDEs for particles, energy, rotation



Particle flux — Particle sources/sinks

Particle density: $$\frac{\partial n_s}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}(r\Gamma_s) = S_s$$

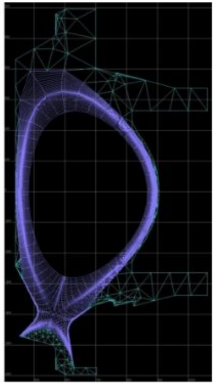
Energy: $$\frac{3}{2}\frac{\partial P_s}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}(r q_s) = Q_s$$

Heat flux — Heat sources/sinks

- Confined hot plasma region where fusion happens
- Computational bottleneck: turbulent fluxes, with highly nonlinear transport coefficients

Multiphysics: magnetic equilibrium, plasma turbulent transport, MHD, heating and fuelling

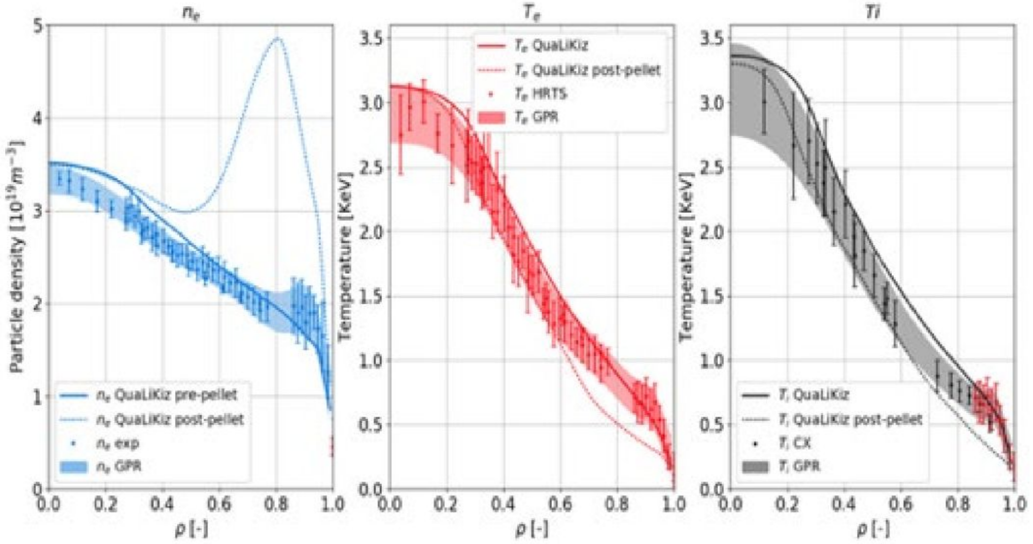
Plasma edge:
2D coupled PDEs for plasmas, 3D kinetic for neutrals

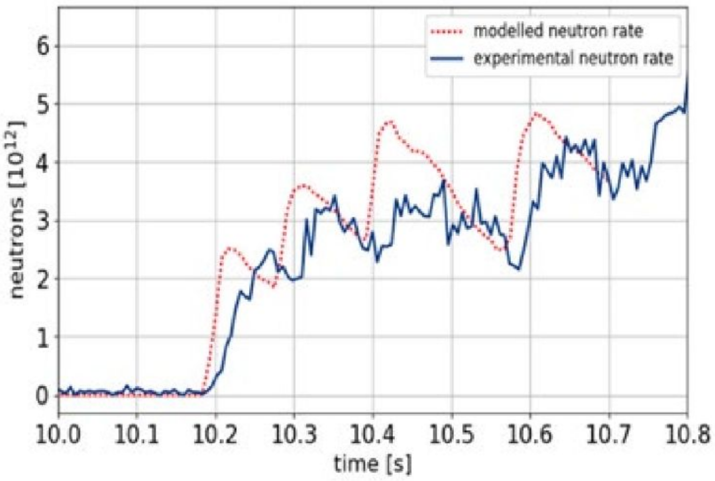


- Plasma fuelling and pumping and impact on confinement
- Impact of plasma exhaust on wall materials

Multiphysics: magnetic equilibrium, plasma + neutral transport, atomic and molecular interactions, plasma material interaction

Example application: understanding physical mechanism of fast isotope mixing in multiple-isotope JET experiments





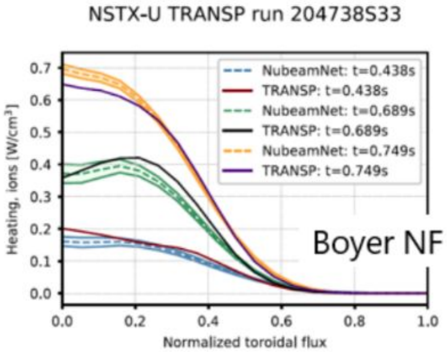
Marin et al Nucl. Fusion 2021

- Multiphysics simulation:
  - Physics-based turbulence model
  - Fueling with injection of frozen deuterium pellets which ablate into initial pure hydrogen plasma
  - Deuterium fusion leads to observed neutrons, measuring timescales of deuterium transport

- Acceptable sim2real gap (not trivial!) but ~1 week per simulation

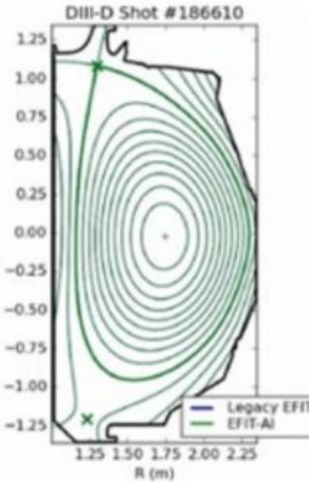
Key method for integrated modelling speed is incorporation of learned ML surrogates of physics components

Neutral Beam Injection and Current Drive



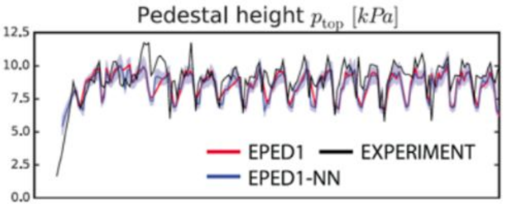
Boyer NF 2019

Magnetic equilibrium



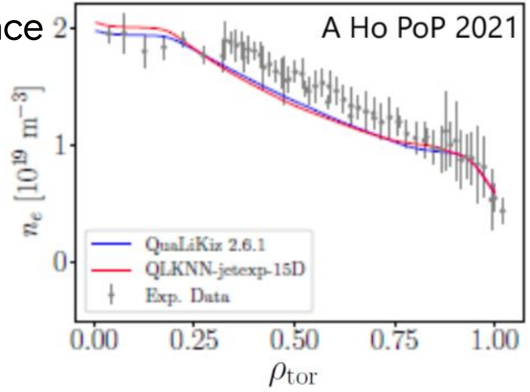
Lao PPCF 2022

Edge Transport Barriers (H-mode pedestal)



Meneghini NF 2017

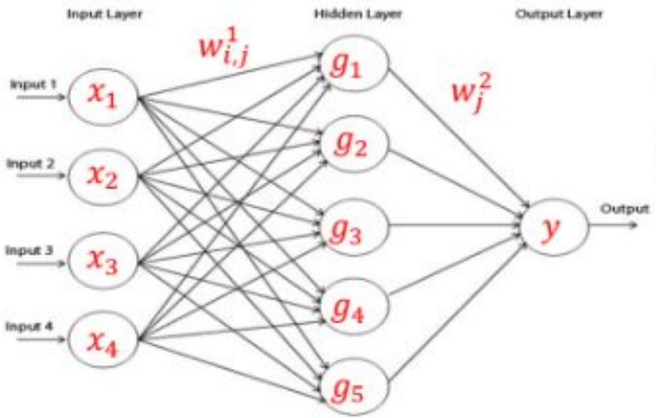
Plasma turbulence



A Ho PoP 2021

Intermezzo on ML-surrogates. Nonlinear regression of model databases

Fully connected feedforward neural network (simple topology)



x: Inputs, e.g. temperature and density gradients, magnetic geometry info
y: Output: e.g. ion heat flux
w: free weights for optimization

$$y = \sum w_j^2 g_j \left(\sum w_{i,j}^1 x_i \right)$$

With, e.g. $g(x) = \tanh(x)$

Normally, optimize weights by minimizing cost function: $C = \sum_N (t_N - y_N)^2 + \lambda \sum (w_{ij})^2$

t_N are target values from generated dataset
λ is the regularization factor. Critical for avoiding overfitting

- Provides an analytical formulas with analytical derivatives
- Complex nested nonlinear functions. Different versions/models can have different topology
- Autodiff is really key for incorporating ML-surrogates into differentiable simulators

TORAX overview

2

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

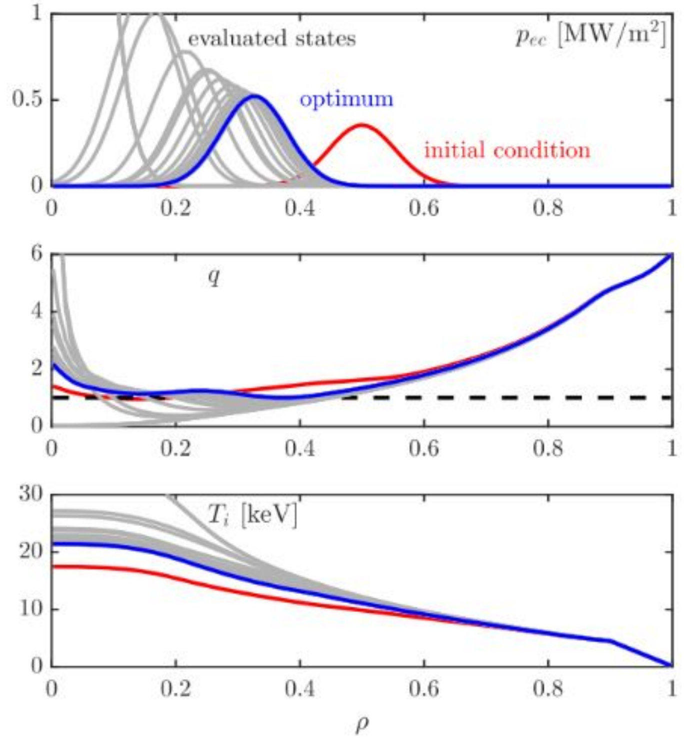
Previous state-of-the-art in tokamak differentiable simulation

RAPTOR control-oriented simulator [Felici PPCF 2012]

MATLAB code, not-auto-differentiable, challenges with scalability and extensibility particularly with ML-surrogates

TORAX auto-differentiatiable with JAX:

- Easy to extend with any analytical models or ML-surrogate
- Easy to extend simulation sensitivity analysis and optimization with new parameter inputs
- ML-surrogate greatly facilitated by JAX's inherent support for neural network development and inference
- Python facilitates coupling within various workflows
- Scalable for large-scale validation and UQ



van Mulders NF 2021
SQP optimization of an ITER scenario

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

jax.readthedocs.io

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(compile-time) ≈ O(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.

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

Ion and electron heat equations

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

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

Electron density equation

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

Current diffusion equation

$$\frac{16\pi^2\sigma_{||}\mu_0\hat{\rho}\Phi_b^2}{F^2}\left(\frac{\partial\psi}{\partial t}-\frac{\hat{\rho}\dot{\Phi}_b}{2\Phi_b}\frac{\partial\psi}{\partial\hat{\rho}}\right)=$$
$$\frac{\partial}{\partial\hat{\rho}}\left(\frac{g_2g_3}{\hat{\rho}}\frac{\partial\psi}{\partial\hat{\rho}}\right)-\frac{8\pi^2V'\mu_0\Phi_b}{F^2}\langle j_{ni}\rangle$$

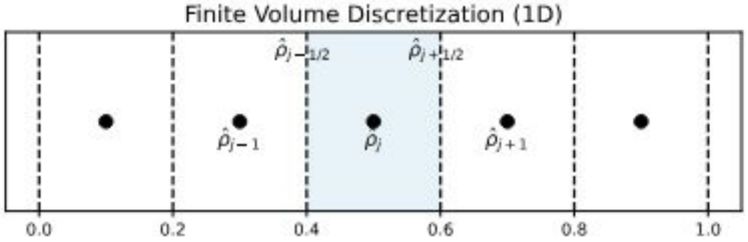
- Moment equations of underlying kinetic equations
  - Toroidal symmetry + flux-surface averaging reduces to 1D
  - Scale-separation of turbulence, sources: "function calls" of reduced order models verified against high-fidelity

- Zero-gradient boundary conditions on axis

- Temperature, density
  - Dirichlet boundary conditions at edge
- ψ (poloidal magnetic flux):
  - Neumann boundary condition at edge (fixed current)
  - or Dirichlet boundary condition for next timestep (edge voltage)

- Any subset of these equations are evolved: non-evolved profiles and initial conditions set in config with user data (xarray, numpy, python primitives)

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



- TORAX JAX fvm package inspired by FiPy[1] and uses similar API
  - Constrained to solving convection-diffusion equations
- For (particle) convection, power-law α-weighting scheme based on Péclet number
- Constructs nonlinear/linear systems of equations for solvers based on solver method

[1] https://www.ctcms.nist.gov/fipy/

Temporal discretization and system composition

Theta method, first-order in time.
θ=1, fully-implicit (default)

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

Governing equations more generally decomposed as follows

$$\frac{3}{2}V'^{-5/3}\left(\frac{\partial}{\partial t}-\frac{\dot{\Phi}_b}{2\Phi_b}\frac{\partial}{\partial\hat{\rho}}\hat{\rho}\right)\left[V'^{5/3}n_iT_i\right]= \frac{1}{V'}\frac{\partial}{\partial\hat{\rho}}\left[\chi_i n_i\frac{g_1}{V'}\frac{\partial T_i}{\partial\hat{\rho}}-g_0q_i^{\mathrm{conv}}T_i\right]+Q_i$$

$$\longrightarrow$$

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

- **x** is state vector, subset of $\{T_i, T_e, n_e, \psi\}$, at time t or t+Δt
- $\tilde{\mathbf{T}}$ 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.
- **c** is vector with source terms + boundary condition terms
- **u** corresponds to all "known" or quantities at time t and t+Δt , e.g. boundary conditions, heating amplitude trajectories, prescribed profiles, etc.

Solver methods

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

- Predictor corrector: fixed point iteration for k (user-defined) steps on $\mathbf{x}_{t+\Delta t}$ in $\tilde{\mathrm{T}}$, $\bar{\mathrm{C}}$, **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 **R** is the LHS-RHS of (1)

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)

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

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 the Jacobian is 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

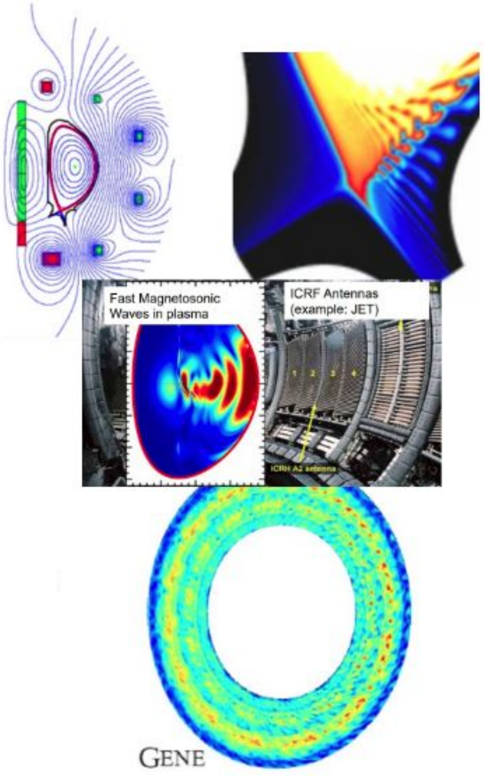
Presently implemented physics models/couplings (non-exhaustive list)

- ML-surrogates where available
  - Turbulence (QLK-NN [van de Plassche 2020, Hamel 2025*], TGLF-NN [Zanisi 2025])
  - Heating (TORIC-NN [Wallace APS 24])

- Fast analytical models where appropriate
  - Bootstrap current, neoclassical transport, ECCD
  - Mavrin polynomial fits to ADAS for line radiation, Bremsstrahlung
  - Fusion power, ion-electron collisional exchange, Ohmic power

- Pre-calculated sequence of geometry inputs: wrappers for CHEASE, FBT, EQDSK. IMAS underway

- Collaborations key
  - TORAX aims to be natural target platform for community-wide ML-surrogates
    - ONNX data storage for portability
  - Design focus on modularity



* https://github.com/google-deepmind/fusion_transport_surrogates (released last Friday)

Runtimes and solver comparison

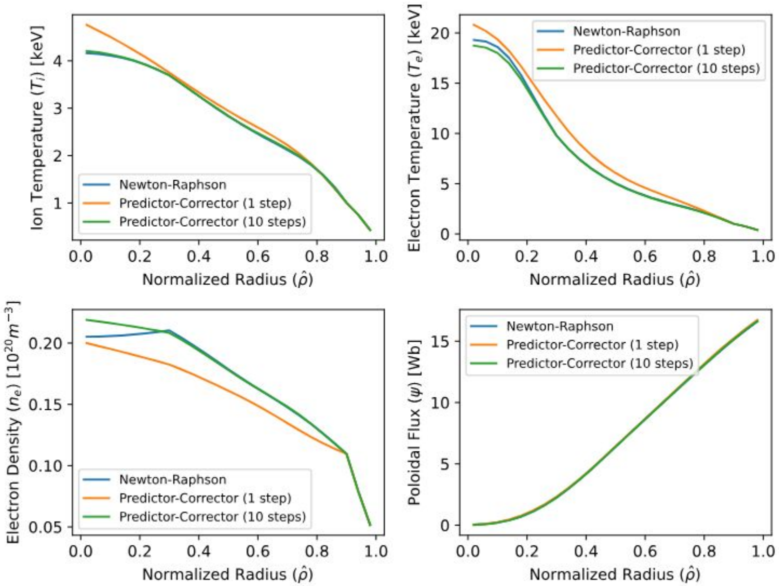


FIG. 5. Comparison of simulated T_i, T_e, n_e, and ψ profiles at t = 10 s for the Newton-Raphson, Predictor-Corrector (1 step), and Predictor-Corrector (10 steps) solvers, using the example iter-hybrid_rampup.py configuration.

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 scenario current ramp-up.

- Faster than realtime for this config
- Still some low hanging fruit for runtime performance optimization

Comparison with other integrated modelling suites

	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 or https://github.com/Cambridge-ICCS/FTorch	Mex-files from Fortran	Python w/JAX
Range of physics models	Broad, high-fidelity, no restriction (apart from speed)	Parameterized: formulas, ML-surrogates	Any: analytical models + ML-surrogates can be in JAX kernel. Non-JAX models can be injected into kernel as explicit terms High-fidelity models can still be coupled into solver kernel: JAX compilation can be disabled. Allows evaluation of ML-surrogates against ground truth within the same framework

Towards full-sim differentiation with Forward Sensitivity Analysis: key is the Jacobian of the PDE system residual

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)

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

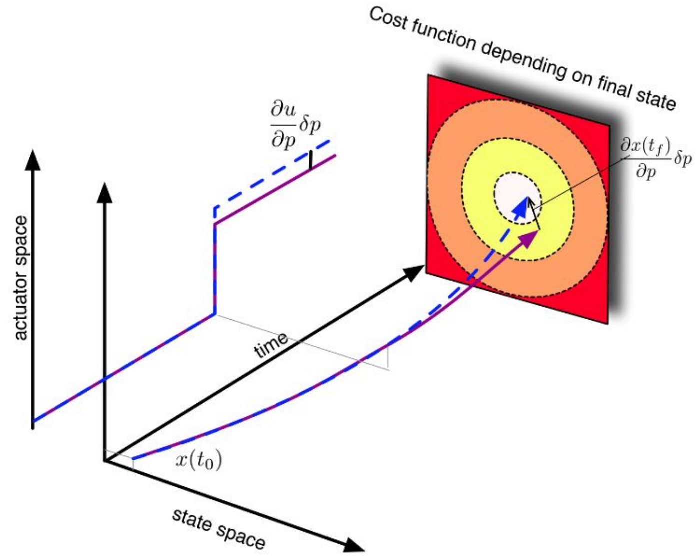
Forward Sensitivity Analysis method

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

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!



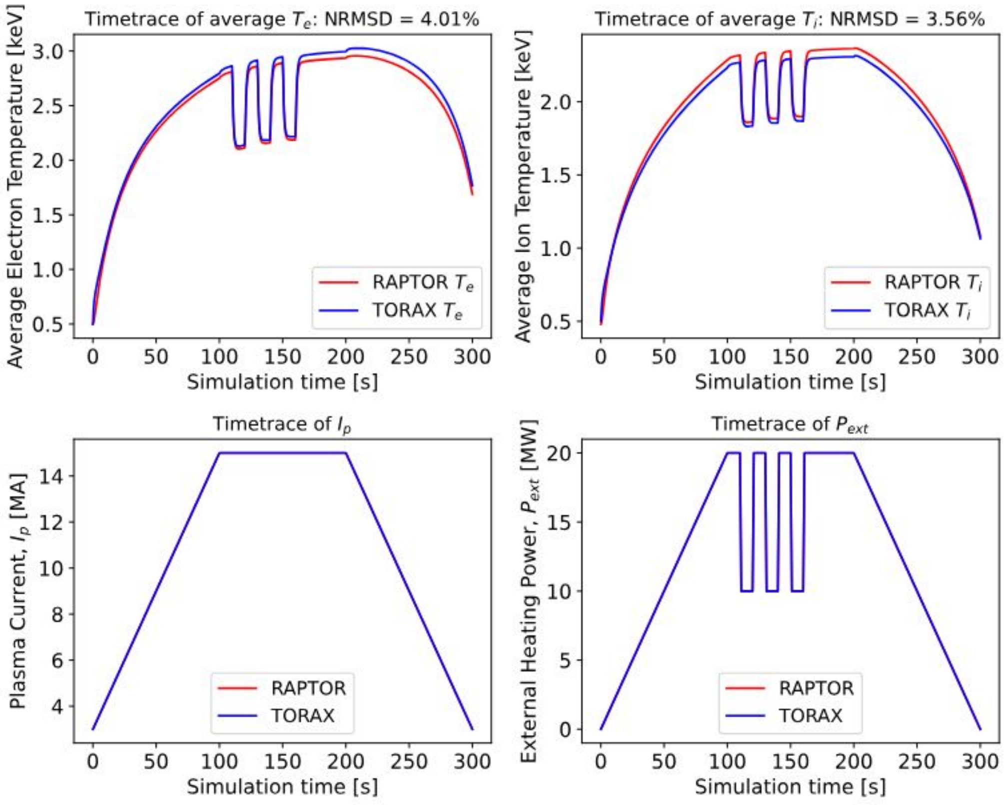
Key tool for sensitivity analysis, data-driven parameter identification, trajectory optimization methods

From Felici PhD 2011 (https://infoscience.epfl.ch/record/168656?ln=en) and Felici PPCF 2012

Benchmarks vs RAPTOR

3

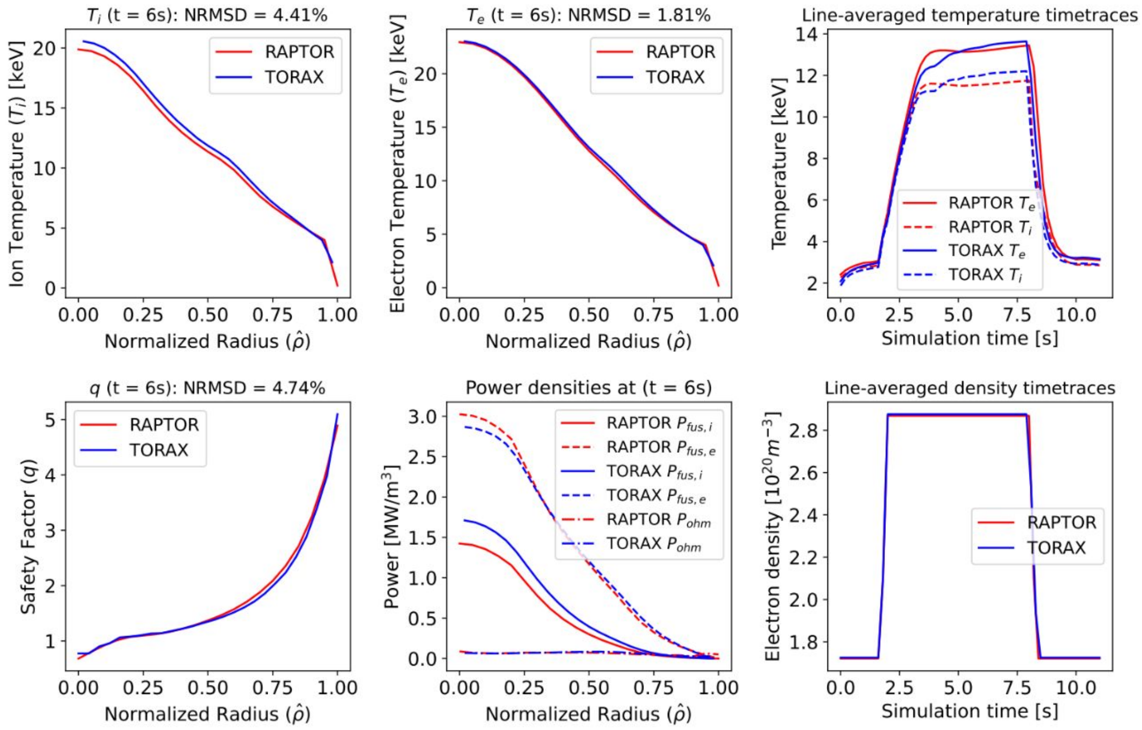
Verification: TORAX vs RAPTOR agreement for ITER-like cases



Modeling settings:

- ITER inspired params
- Nonlinear Newton-Raphson solver
- Heat transport + current diffusion
- 20MW heating: modulated
- Constant transport coefficients

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



Modeling settings:

- SPARC full-pulse scenario
- Nonlinear Newton-Raphson solver
- Heat transport + current diffusion
- Sequence of magnetic equilibria
- 11MW heating power
- ML-surrogate turbulent transport model (QLKNN10D*)

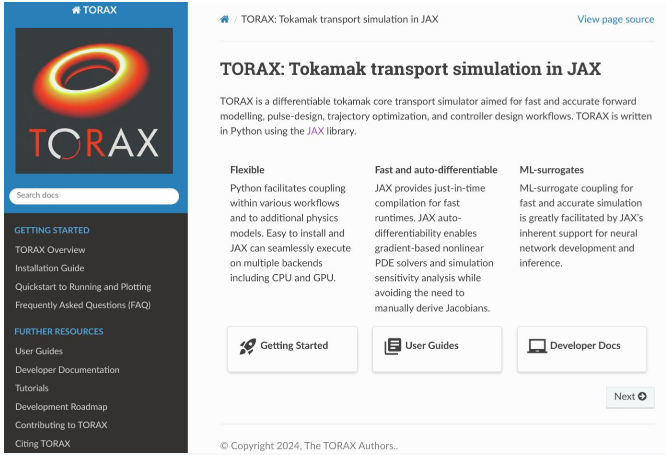
Δt=0.2s: RAPTOR walltime: ~70s , TORAX walltime: ~14s

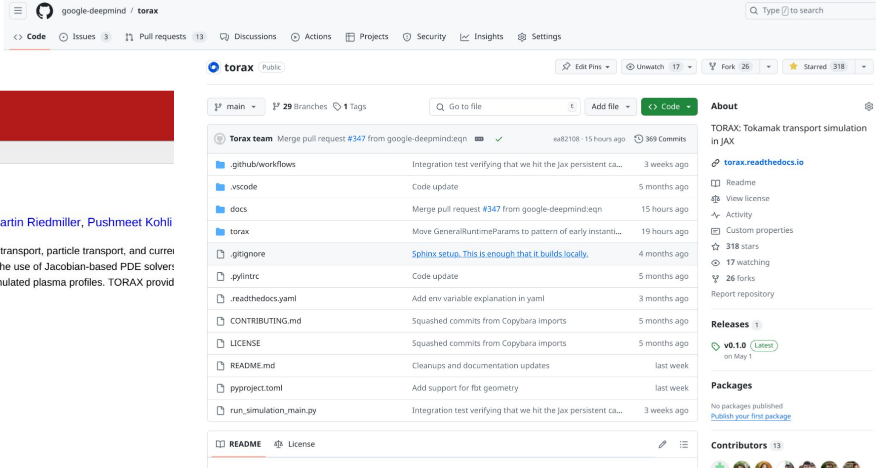
* K.L. van de Plassche et al., PoP 2020

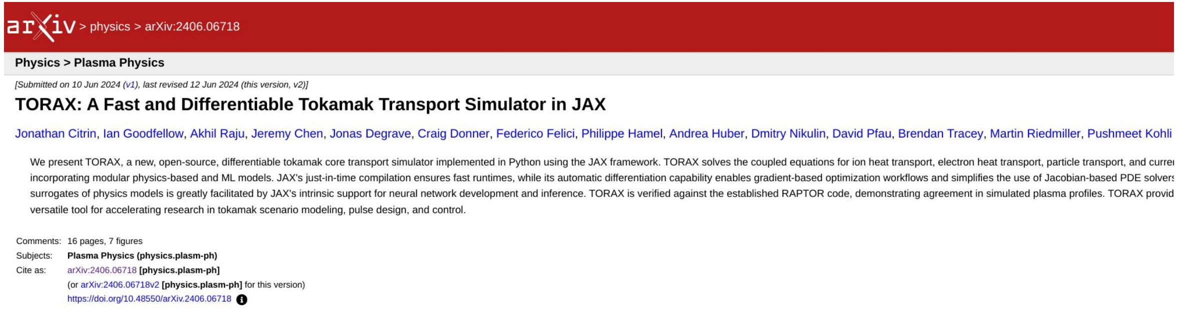
TORAX open-source with permissive Apache 2.0 license

- 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







Outlook

4

Roadmap: towards higher physics fidelity and pulse planning applications

Short term developments

- Reach physics feature parity and beyond compared to SOTA control-oriented tokamak sim
  - Sawteeth
  - Rotation + ExB shear
- Initial applications for tokamak pulse planning with collaborators

Priorities for improved physics + numerics

- Multi-ion + impurity transport
- Improved ML-surrogates
  - Turbulence
  - Pedestal
  - Magnetic equilibrium
  - Heat/**particle sources**
  - **Plasma edge + wall**
- Demonstrate gradient-driven optimization applications
- Demonstrate large-scale batch simulations on GPU for optimization and UQ

Validation/calibration against data

- Open source datasets for integrated modelling validation
  - Engagement with fusion community
- Data can also be used to further calibrate models, using gradient-driven methods

Roadmap: initial applications and collaborations underway

CFS MOSAIC Pulse Planning Workflow:

- GSPulse [Wai APS24]. Optimizes coil currents over full target pulse trajectory
- TORAX for internal plasma dynamics

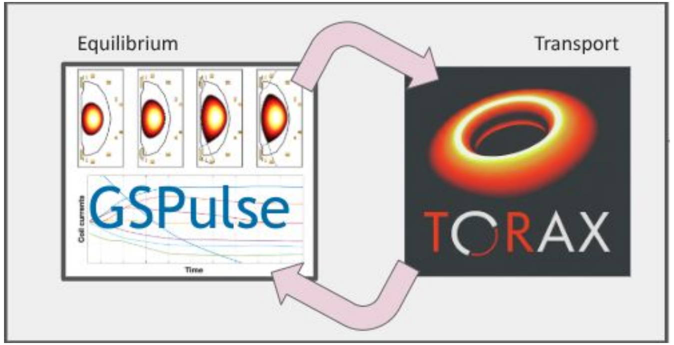


Image from Devon Battaglia

ITER Pulse Design Simulator w/CEA + Ignition Computing

- Coupling to IMAS underway [Schneider, Bellourd, Sanders, van Vugt]

UKAEA

- Progress in preparing TORAX for STEP simulation and benchmarking with JETTO [T. Brown, L. Zanisi]

CEA

- E. Stancar for WEST trajectory optimization

Summary

- TORAX, a new Python-based tokamak core transport code
  - With JAX: Fast for many-query applications, and auto-differentiable
  - Targeted for easy coupling to range of ML-surrogates
  - Verified against previous SOTA tokamak control-oriented simulator, RAPTOR

- Open-source for wider community impact
  - Supporting applications
  - Supporting integration of new physics models and ML-surrogates
  - Open source data for validation + calibration; 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