Applying Differentiable Programming to Experimental, Theoretical, and Computational Plasma Physics

A. S. Joglekar[1,2]

[1]Ergodic LLC., Seattle, WA

[2]Laboratory for Laser Energetics, University of Rochester, Rochester, NY

Collaborators & Acknowledgments

A. G. R. Thomas

A. Milder, K. Miller, J. Palastro, R. Follett, D. Froula

This material is based upon work supported by the DOE Office of Fusion Energy under Award Number(s) DE-SC0024863, NERSC award FES-ERCAP0026741, DOE NNSA under Award Number DE-NA0003856

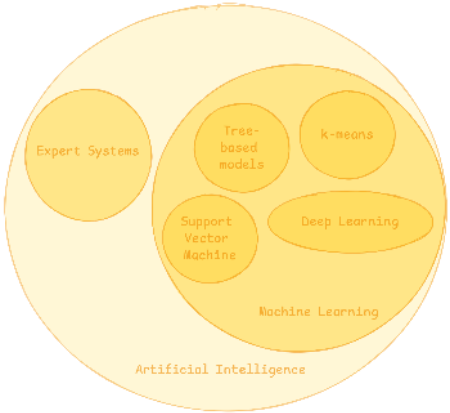
By writing algorithms using a differentiable framework, we can tackle the usual plasma physics problems using a 'novel' approach

- Describe and motivate differentiable programming
- Apply it to discover new kinetic physics
- Apply it differently to develop reduced models for fluid codes to reproduce the new kinetic physics

This paradigm allows us to leverage the $\mathcal{O}(\$B)$ investments in numerical frameworks and HPC for ML from the tech companies

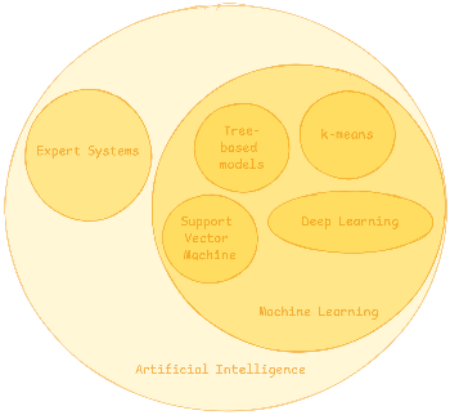
Modern AI is primarily based on deep learning

- Artificial Intelligence is when algorithms can accomplish tasks with human-level performance



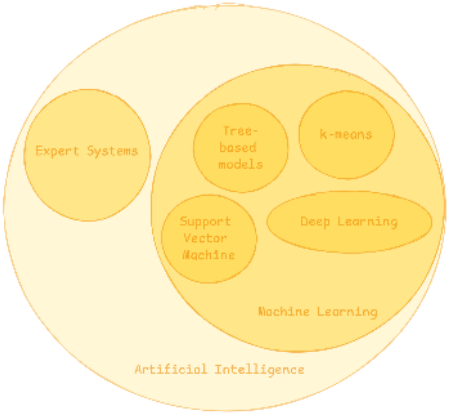
Modern AI is primarily based on deep learning

- Artificial Intelligence is when algorithms can accomplish tasks with human-level performance
- Machine Learning refers to algorithms which do not need to be explicitly programmed but instead, learn from data

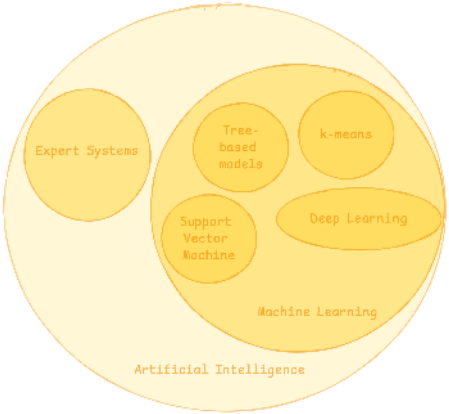


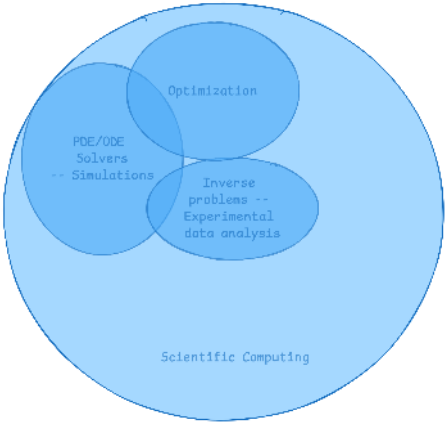
Modern AI is primarily based on deep learning

- Artificial Intelligence is when algorithms can accomplish tasks with human-level performance
- Machine Learning refers to algorithms which do not need to be explicitly programmed but instead, learn from data
- Deep Learning refers to the case where the ML model is a neural network



The bulk of computational plasma physics is contained within scientific computing while AI seems to offer bespoke solutions

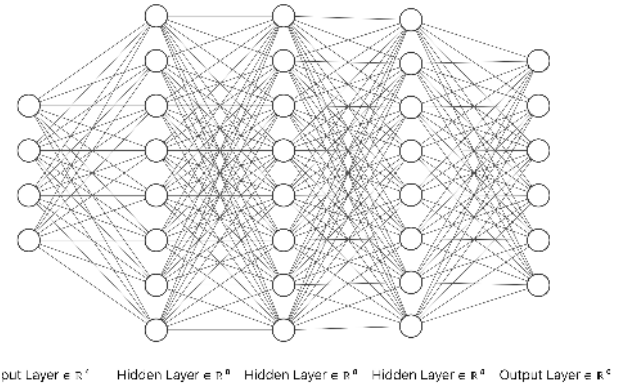




The technology underlying the training of neural networks offers a path for unifying scientific computing and AI

$$\vec{y} = f(\vec{x}) \qquad \hat{\vec{y}} = g(\vec{x}; \theta)$$
$$\hat{\vec{y}} = A_4 \circ A_3 \circ A_2 \circ A_1(\vec{x}; \theta_1)$$

$$A_1 : \mathbb{R}^4 \to \mathbb{R}^8 \qquad A_2 : \mathbb{R}^8 \to \mathbb{R}^8$$
$$A_3 : \mathbb{R}^8 \to \mathbb{R}^8 \qquad A_4 : \mathbb{R}^8 \to \mathbb{R}^6$$



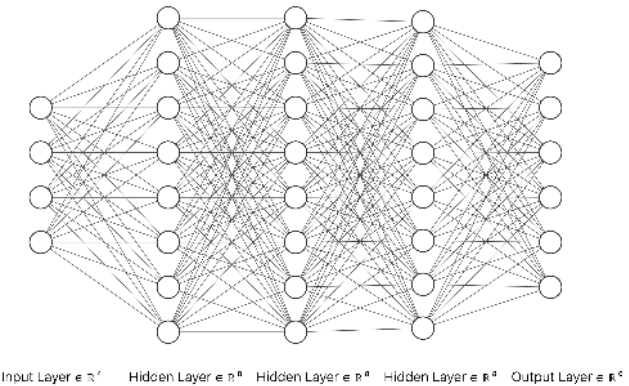
The technology underlying the training of neural networks offers a path for unifying scientific computing and AI

$$\vec{y} = f(\vec{x}) \qquad \hat{\vec{y}} = g(\vec{x};\theta)$$
$$\hat{\vec{y}} = A_4 \circ A_3 \circ A_2 \circ A_1(\vec{x};\theta_1)$$

$$A_1 : \mathbb{R}^4 \to \mathbb{R}^8 \qquad A_2 : \mathbb{R}^8 \to \mathbb{R}^8$$
$$A_3 : \mathbb{R}^8 \to \mathbb{R}^8 \qquad A_4 : \mathbb{R}^8 \to \mathbb{R}^6$$

$$\text{argmin } \mathcal{L}(\theta) : \mathcal{L}(\theta) = \sum [g(\vec{x};\theta) - f(\vec{x})]^2$$

$$\theta^{n+1} = \theta^n + \Delta\theta \frac{\partial L}{\partial \theta}$$



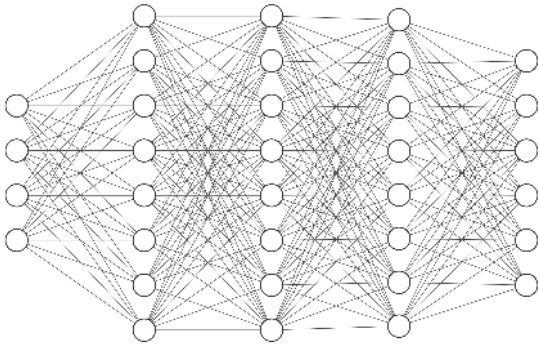
The technology underlying the training of neural networks offers a path for unifying scientific computing and AI

$$\vec{y} = f(\vec{x}) \qquad \hat{\vec{y}} = g(\vec{x};\theta)$$
$$\hat{\vec{y}} = A_4 \circ A_3 \circ A_2 \circ A_1(\vec{x};\theta_1)$$

$$A_1 : \mathbb{R}^4 \to \mathbb{R}^8 \qquad A_2 : \mathbb{R}^8 \to \mathbb{R}^8$$
$$A_3 : \mathbb{R}^8 \to \mathbb{R}^8 \qquad A_4 : \mathbb{R}^8 \to \mathbb{R}^6$$

$$\text{argmin } \mathcal{L}(\theta) : \mathcal{L}(\theta) = \sum [g(\vec{x};\theta) - f(\vec{x})]^2$$

$$\theta^{n+1} = \theta^n + \Delta\theta \frac{\partial L}{\partial \theta}$$



How do you get the gradient?

Symbolic and finite differentiation cannot efficiently calculate gradients of arbitrarily complex numerical programs of $\mathcal{O}(100 - 10^9)$ parameters

Symbolic and finite differentiation cannot efficiently calculate gradients of arbitrarily complex numerical programs of $\mathcal{O}(100 - 10^9)$ parameters

Symbolic differentiation

- Requires a closed-form expression that gets differentiated. Intractable with arbitrarily complex numerical programs because of equation bloat.
- Not efficient

Symbolic and finite differentiation cannot efficiently calculate gradients of arbitrarily complex numerical programs of $\mathcal{O}(100 - 10^9)$ parameters

Symbolic differentiation

- Requires a closed-form expression that gets differentiated. Intractable with arbitrarily complex numerical programs because of equation bloat.
- Not efficient

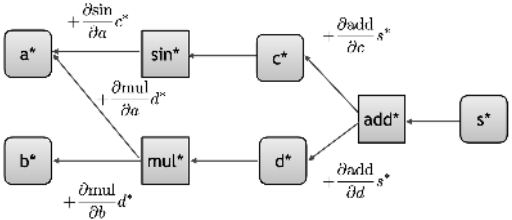
Finite Differentiation

- Susceptible to truncation errors (2nd order, 4th order, and so on)
- Slow! Has to be done once per parameter
- Slightly more efficient, not accurate

Automatic Differentiation is "chain-ruled symbolic differentiation"

- Mathematical primitives have defined derivatives
- Code composed with these primitives has derivatives because of the chain rule
- Algorithm-level modification - Often also called "Algorithmic Differentiation"
- Efficient and Accurate

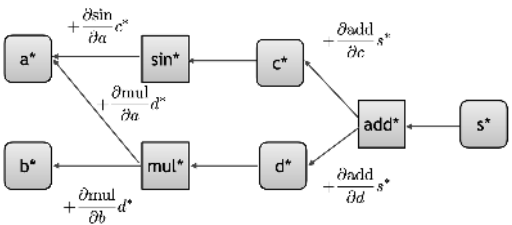
- **Example:** $s(a, b) = \sin(a) + ab$



Automatic Differentiation is "chain-ruled symbolic differentiation"

- Mathematical primitives have defined derivatives
- Code composed with these primitives has derivatives because of the chain rule
- Algorithm-level modification - Often also called "Algorithmic Differentiation"
- Efficient and Accurate

- **Example:** $s(a, b) = \sin(a) + ab$

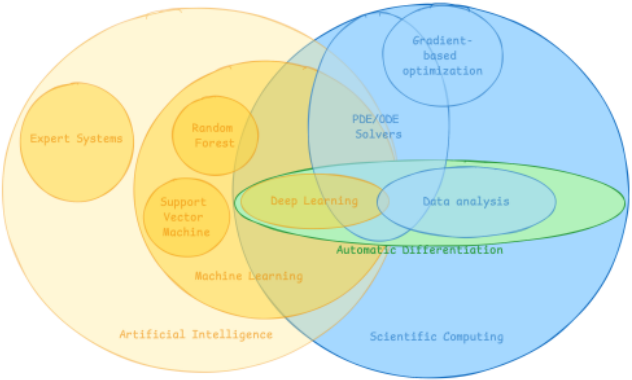


```
from jax import grad
import jax.numpy as jnp

def tanh(x):  # Define a function
  y = jnp.exp(-2.0 * x)
  return (1.0 - y) / (1.0 + y)

grad_tanh = grad(tanh)  # Obtain its gradient function
print(grad_tanh(1.0))   # Evaluate it at x = 1.0
# prints 0.4199743
```

Automatic Differentiation (AD) is intimately related to Deep Learning AND Scientific Computing

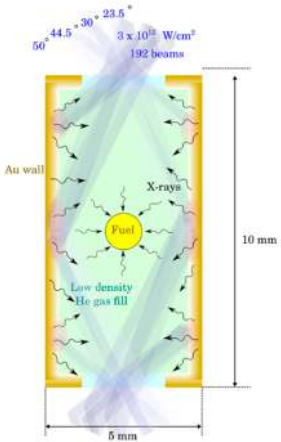


We integrate Deep Learning and Automatic Differentiation with general-purpose Numerical Programming like PDE solvers and Optimization

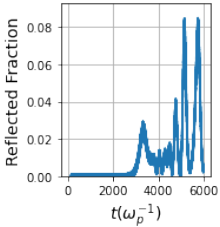
Part 1: Discovering exotic nonlinear kinetic dynamics[1]

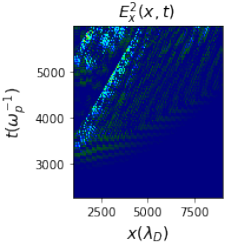
[1] Joglekar and Thomas 2022, *Journal of Plasma Physics*.

Nonlinear electron plasma waves can be excited in inertial fusion experiments

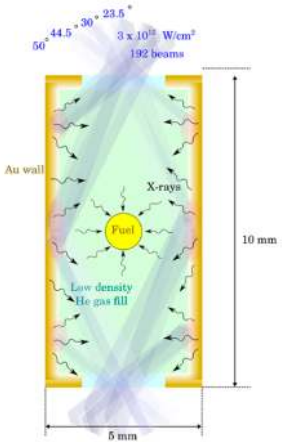


- In Stimulated Raman Scattering (SRS), crossing light waves drive electrostatic waves in a plasma

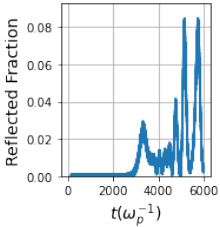


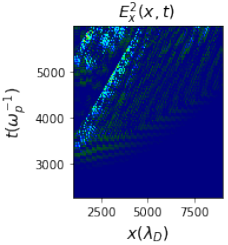


Nonlinear electron plasma waves can be excited in inertial fusion experiments

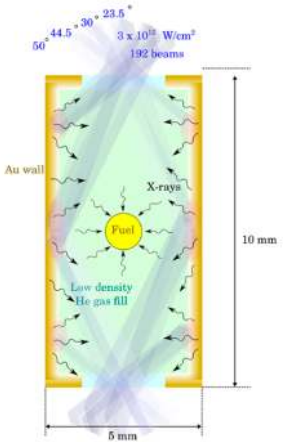


- In Stimulated Raman Scattering (SRS), crossing light waves drive electrostatic waves in a plasma
- After some time, SRS enters a bursty, high reflectivity regime

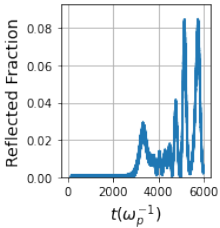


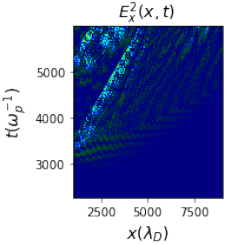


Nonlinear electron plasma waves can be excited in inertial fusion experiments

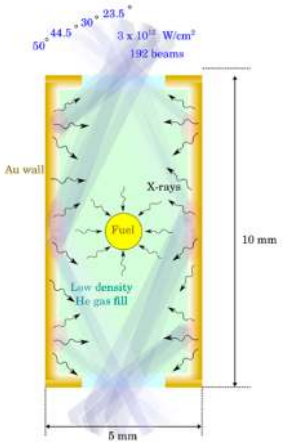


- In Stimulated Raman Scattering (SRS), crossing light waves drive electrostatic waves in a plasma
- After some time, SRS enters a bursty, high reflectivity regime
- This state is correlated with particle trapping caused by the growth in the electrostatic potential

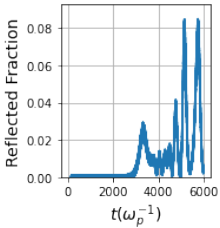


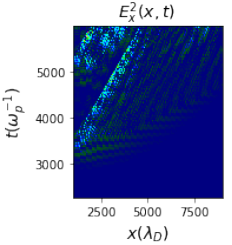


Nonlinear electron plasma waves can be excited in inertial fusion experiments

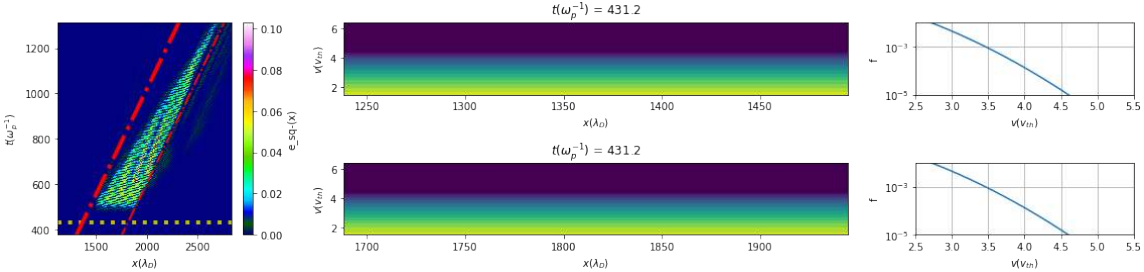


- In Stimulated Raman Scattering (SRS), crossing light waves drive electrostatic waves in a plasma
- After some time, SRS enters a bursty, high reflectivity regime
- This state is correlated with particle trapping caused by the growth in the electrostatic potential
- During this stage, large amplitude wavepackets of $\mathcal{O}(10^2)\lambda_D$ in length are excited

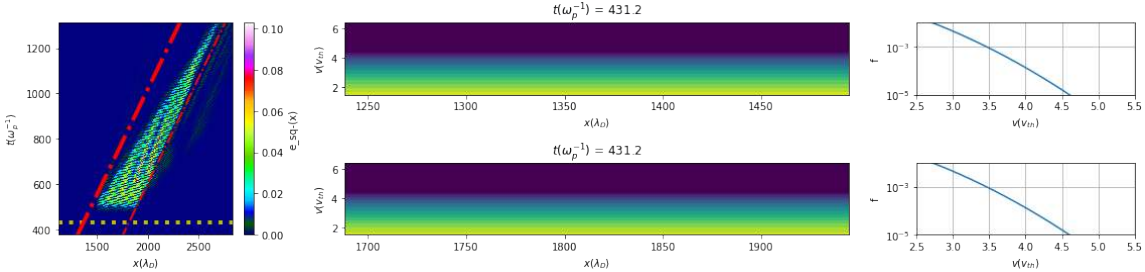




Wavepacket Etching in finite-length non-linear electron plasma waves

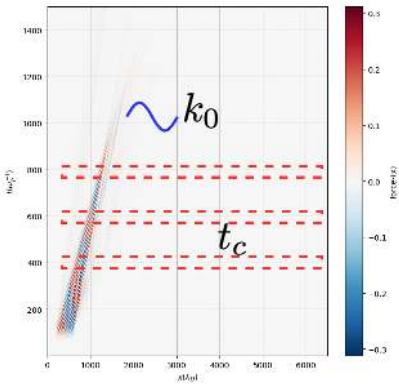


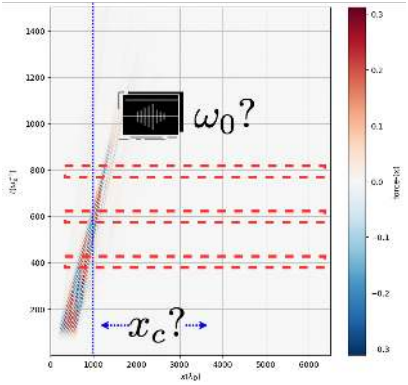
- The distribution function in the back of the wavepacket is nearly Maxwellian due to detrapping



- The distribution function in the back of the wavepacket is nearly Maxwellian due to detrapping
- This causes Landau damping to resume and the back of the wavepacket damps at a faster rate than the freely propagating front

What happens when a second wave is driven to hit the escaped electrons *just right*?





- Holding the width and duration of the driver fixed, the parameters to search over are k, t_c, ω, x_c i.e. the wavenumber, and frequency, and the location in time and in space

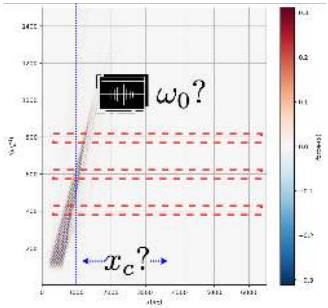
Using a differentiable solver, a neural network can be trained to identify driver conditions that excite nonlinear physics

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$E_T = E + E_d(\vec{p}(\theta))$$
$$\partial_x E = 1 - n_e$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$

mlflow

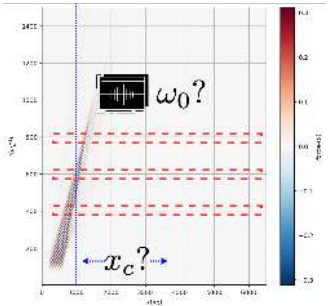
Using a differentiable solver, a neural network can be trained to identify driver conditions that excite nonlinear physics

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$E_T = E + E_d(\vec{p}(\theta))$$
$$\partial_x E = 1 - n_e$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$



Using a differentiable solver, a neural network can be trained to identify driver conditions that excite nonlinear physics

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$E_T = E + E_d(\vec{p}(\theta))$$
$$\partial_x E = 1 - n_e$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$

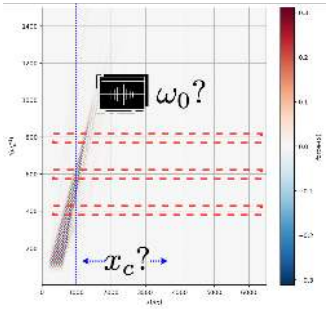


The Task

For $k_0 \in [0.26, 0.27, \ldots, 0.32]$ and $t_c \in [400, 500, \ldots, 800]$, we wish to learn a continuous function for x_c and ω_0

Using a differentiable solver, a neural network can be trained to identify driver conditions that excite nonlinear physics

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$E_T = E + E_d(\vec{p}(\theta))$$
$$\partial_x E = 1 - n_e$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$



The Task

For $k_0 \in [0.26, 0.27, \ldots, 0.32]$ and $t_c \in [400, 500, \ldots, 800]$, we wish to learn a continuous function for x_c and ω_0

The Objective Function

$$\mathcal{L}(\vec{p}(\theta)) = -\sum_{1000}^{1200} \Delta t \sum_{\text{box}} \Delta x \left[E_x^2(t, x; p(\theta)) + \sum_v \Delta v\ (f \log(f) - f_{\text{MX}} \log f_{\text{MX}}) \right]$$

The algorithm finds a place to put the second wave to minimize damping

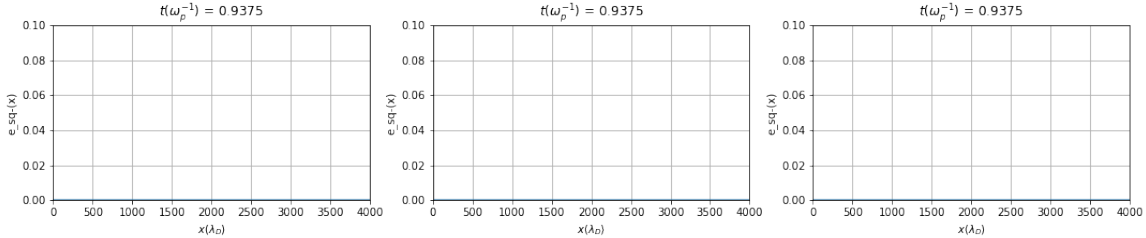
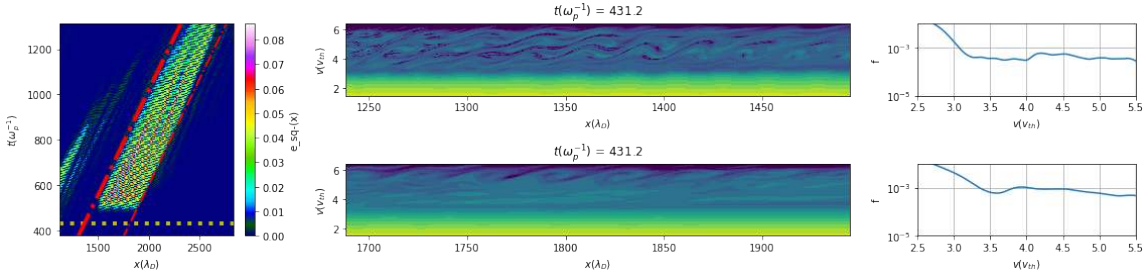


Figure: Left: 1st Pulse, Middle: 2nd Pulse, Right: Both

- The middle frame dissipates earlier than the frame on the right
- The right frame is NOT the superposition of the content of the first and second frame

Further interrogation reveals novel kinetic, nonlinear dynamics



- The incoming streaming electrons maintain the flattening of the distribution function
- The flattening of the distribution function prevents the Landau damping at the back of the wavepacket

Part 2: Reduced models for hydro codes - Fluid modeling of Part 1[2]

[2] Joglekar and Thomas 2023, *IOP Machine Learning: Science & Technology*.

To model Part 1 using a fluid model, we need to add kinetic effects

Say we want to model linear and nonlinear Landau damping using a fluid code

Kinetic Description

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2\partial_v f)$$

Fluid Description

Poisson-Electrostatic Solver

$$E_T = E + E_d(\vec{p})$$
$$\partial_x E = 1 - n_e$$

To model Part 1 using a fluid model, we need to add kinetic effects

Say we want to model linear and nonlinear Landau damping using a fluid code

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$

$$\partial_t n + \partial_x(u\ n) = 0,$$
$$n\,(\partial_t u + u\ \partial_x u) = -\partial_x p + nE_T,$$
$$p = n^\gamma,$$

$$E_T = E + E_d(\vec{p})$$
$$\partial_x E = 1 - n_e$$

This model does not contain Landau damping

To model Part 1 using a fluid model, we need to add kinetic effects

Say we want to model linear and nonlinear Landau damping using a fluid code

Kinetic Description

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2 \partial_v f)$$

Fluid Description

$$\partial_t n + \partial_x(u\ n) = 0,$$
$$n\,(\partial_t u + u\ \partial_x u) = -\partial_x p + nE_T + 2\ n\ \nu_L * u,$$
$$p = n^\gamma,$$

Poisson Electrostatic Coupling

$$E_T = E + E_d(\vec{p})$$
$$\partial_x E = 1 - n_e$$

This model does not contain nonlinear Landau damping

To model Part 1 using a fluid model, we need to add kinetic effects

Kinetic Description

$$\partial_t f + v\ \partial_x f + E_T\ \partial_v f = C_{ee}(f)$$
$$C_{ee}(f) = \nu_{ee}\partial_v(vf + v_0^2\partial_v f)$$

Relevant Electrostatic Scales

$$E_T = E + E_d(\vec{p})$$
$$\partial_x E = 1 - n_e$$

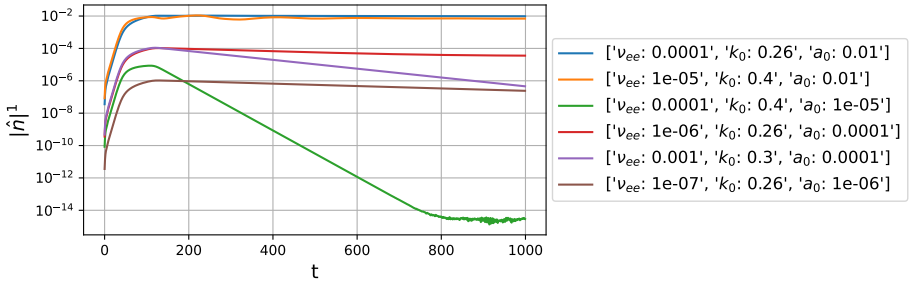
Fluid Description

$$\partial_t n + \partial_x(u\ n) = 0,$$
$$n\,(\partial_t u + u\ \partial_x u) = -\partial_x p + nE_T + 2\ n\ \frac{\nu_L * u}{1+\delta^2},$$
$$p = n^\gamma,$$
$$\partial_t \delta_k = v_{ph}\ \partial_x \delta + \nu_g\ \frac{|E * \nu_L|}{1+\delta^2},$$
$$\nu_g = 10^{f(\tilde{E}_k, \tilde{\nu}_{ee}, \tilde{k}; \theta_g)},$$
$$f(E, \nu_{ee}, k; \theta) = 3\tanh\left(g(\tilde{\nu}_{ee}, \tilde{k}, |\hat{\tilde{E}}^k|; \theta)\right)$$

This model has a quantity that represents trapped particles and can do both [a]

[a] Divol 2003, Tran 2020

A dataset of single-mode, weakly-collisional, nonlinear Landau damping simulations is acquired using a Vlasov-Fokker-Planck code



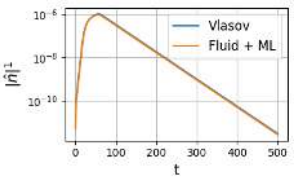
The training data are simulations with parameters given by

$$\nu_{ee} \in [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}],$$
$$k \in [0.26, 0.28, 0.3, 0.32, 0.34, 0.36, 0.38, 0.4],$$
$$a_0 \in [10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}].$$

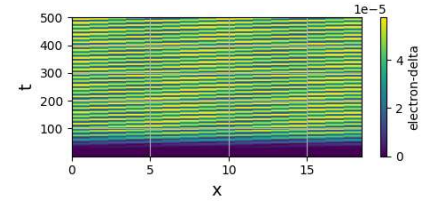
The ML-enhanced fluid model reproduces single-mode NL-EPW

(a)Table stakes - recover non-ML limit

(b) $\delta(t, x)$ stays small



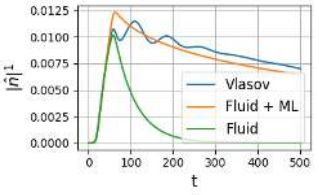
(a) Linear Landau damping



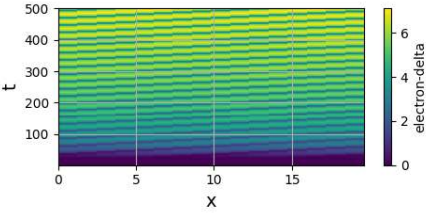
(b) $\delta(t, x)$ shows no trapping

(c) Fluid model damps too fast at the linear value, Fluid + ML recovers Vlasov damping rate

(d) $\delta(t, x)$ grows to ~ 7



(c) Nonlinear Landau damping

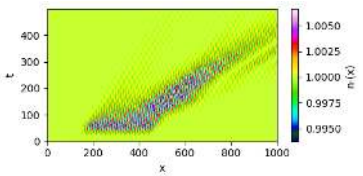


(d) $\delta(t, x)$ is non-zero indicating trapping effects are active

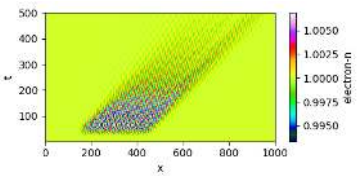
The model is geometry-agnostic and therefore, reproduces finite length behavior without further training

(c) The nonlocal, δ-based model does reproduce etching

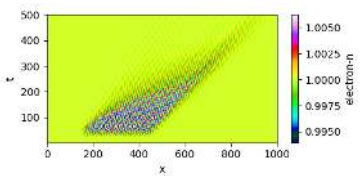
(d) shows that $\delta(t, x)$ grows to account for the hot electron effects and the spatiotemporally nonlocal suppression of Landau damping



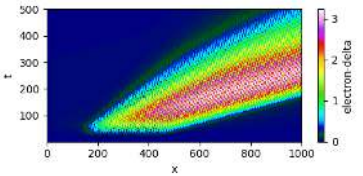
(a) Vlasov



(b) Fluid + Local Model



(c) Fluid + $\delta(t, x)$



(d) $\delta(t, x)$

By writing numerical algorithms using a differentiable framework, we can tackle plasma physics problems using an inverse modeling approach

- Many problems in plasma physics can be framed as optimization problems
- Modern ML frameworks facilitate general purpose numerical programs that are differentiable
- Writing differentiable programs allows the use of gradient-based techniques to solve those problems
- It also puts neural networks and mathematical modeling on equal footing enabling seamless mixing and matching
- We discussed examples of using this paradigm to
  - Develop Theory - identify novel nonlinear, kinetic plasma behavior
  - Improve Computational Methods - Train reduced models for hydro codes
- We leverage $\mathcal{O}(\$B)$ investments from tech companies to develop novel methods for plasma physics research