Message-passing neural PDE solvers

Richard Strunk, 7. May 2024

Disclaimer

- I’m not an expert on differential equations

Disclaimer

- I’m not an expert on differential equations
- I could not independently verify runtime & MSE claims

Disclaimer

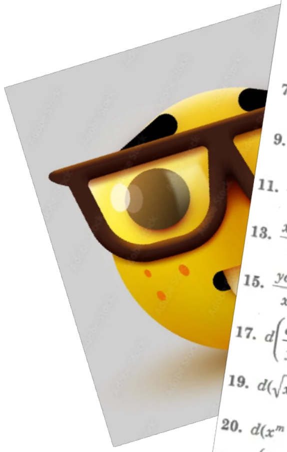
- I’m not an expert on differential equations
- I could not independently verify runtime & MSE claims
- Their implementation differs from the paper in a lot of details

Differential equations...

$$\nabla^2 u = \frac{2m}{\hbar^2}[E - V(x, y, \ldots)]u = 0$$

$$\nabla^2 u - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} + \lambda^2 u = 0$$

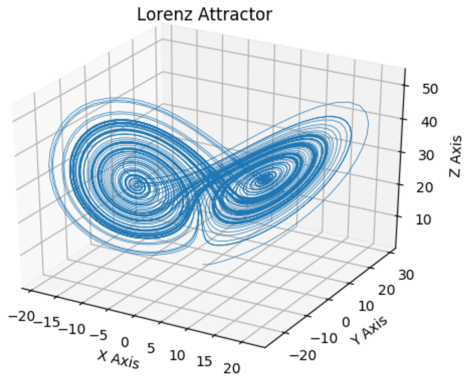
$$\nabla^2 u = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$$



1. $xdy + ydx = d(xy)$
2. $d(x+y) = dx + dy$
3. $d\left(\frac{y}{x}\right) = \frac{x\,dy - y\,dx}{x^2}$
4. $d\left(\frac{x}{y}\right) = \frac{y\,dx - x\,dy}{y^2}$
5. $d\left(\frac{x^2}{y}\right) = \frac{2xy\,dx - x^2\,dy}{y^2}$
6. $d\left(\frac{y^2}{x}\right) = \frac{2xy\,dy - y^2\,dx}{x^2}$
7. $d\left(\frac{x^2}{y^2}\right) = \frac{2xy^2\,dx - 2x^2y\,dy}{y^4}$
8. $d\left(\frac{y^2}{x^2}\right) = \frac{2x^2y\,dy - 2xy^2\,dx}{x^4}$
9. $\frac{xdy + ydx}{xy} = d(\log xy)$
10. $\frac{ydx - xdy}{xy} = d\left(\log \frac{x}{y}\right)$
11. $\frac{xdy - ydx}{xy} = d\left(\log \frac{y}{x}\right)$
12. $\frac{dx + dy}{x + y} = d\log(x+y)$
13. $\frac{xdx + ydy}{x^2 + y^2} = d\left(\log\sqrt{x^2+y^2}\right)$
14. $\frac{xdy - ydx}{x^2 + y^2} = d\left(\tan^{-1}\frac{y}{x}\right)$
15. $\frac{ydx - xdy}{x^2 + y^2} = d\left(\tan^{-1}\frac{x}{y}\right)$
16. $d\left(\frac{-1}{xy}\right) = \frac{xdy + ydx}{x^2y^2}$
17. $d\left(\frac{e^x}{y}\right) = \frac{ye^x dy - e^x dy}{y^2}$
18. $d\left(\frac{e^y}{x}\right) = \frac{xe^y dy - e^y dx}{x^2}$
19. $d(\sqrt{x^2+y^2}) = \frac{xdx + ydy}{\sqrt{x^2+y^2}}$
20. $d(x^m\,y^n) = x^{m-1}\cdot y^{n-1}(mydx + nxdy)$
21. $d\left(\frac{1}{2}\log\frac{x+y}{x-y}\right) = \frac{xdy - ydx}{x^2 - y^2}$
22. $\frac{d[f(x,y)]^{1-n}}{1-n} = \frac{f'(x,y)}{[f(x,y)]^n}$
23. $d\left(\frac{1}{y} - \frac{1}{x}\right) = d\left(\frac{1}{y}\right) - d\left(\frac{1}{x}\right) = \frac{dx}{x^2} - \frac{dy}{y^2}$

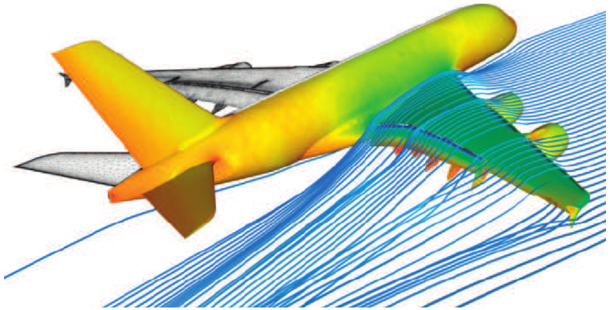
..actually important?

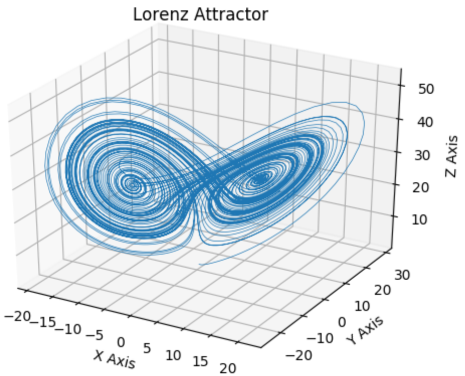
Fluids, waves, wind



..actually important?

Fluids, waves, wind

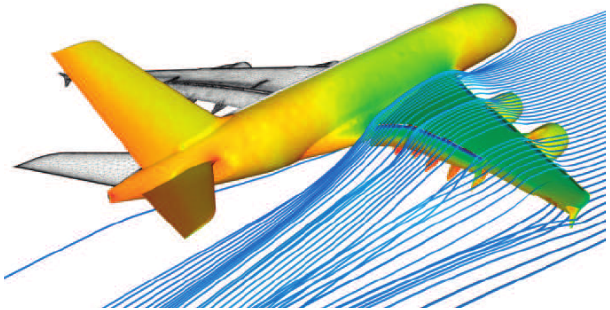


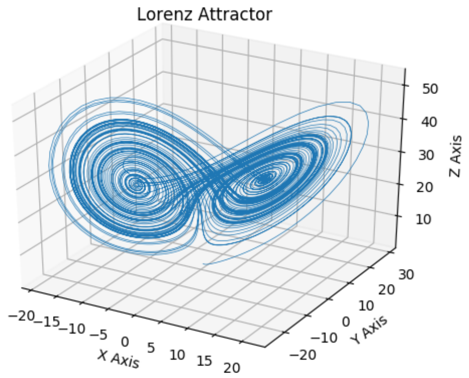


..actually important?

Fluids, waves, wind

Thermal conduction

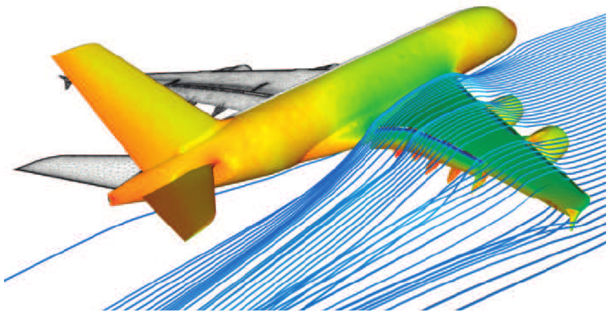


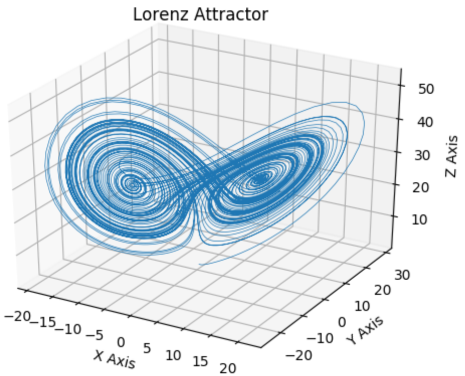


..actually important?

Fluids, waves, wind

Thermal conduction



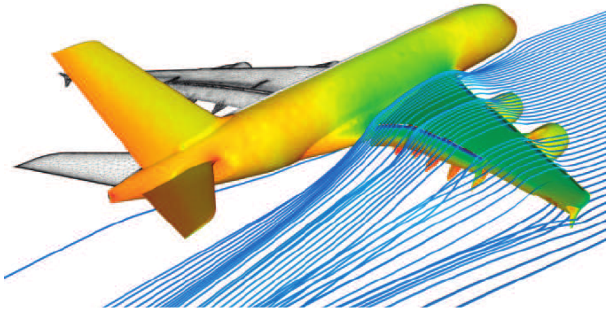


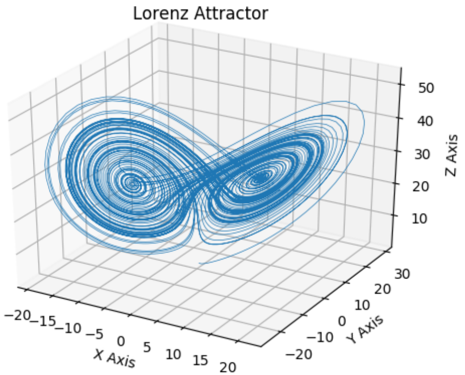
..actually important?

Fluids, waves, wind

Thermal conduction

Movements of stars





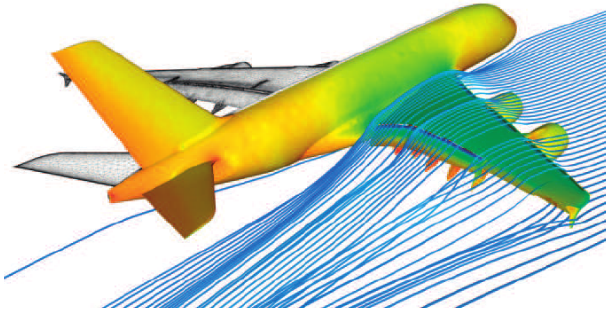
..actually important?

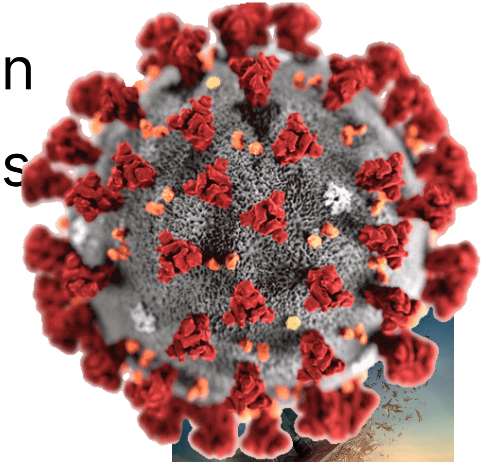
Fluids, waves, air

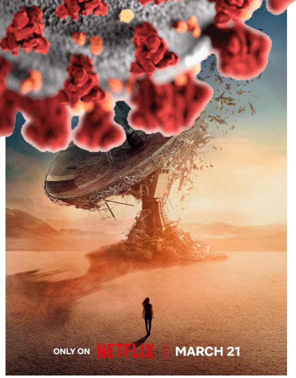
Thermal conduction

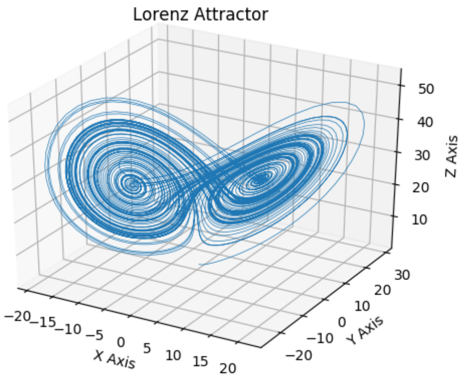
Movements of stars

Epidemiology..









Learning entry

- Absolutely not this paper

Learning entry

- Absolutely not this paper
- https://github.com/barbagroup/CFDPython (→ /w Python Code)
- 3B1B, Differential Equations (→ High quality)
- „PDE Strauss“ (→ Free on Google Books)

Partial differential equations

$$x = [x_1, x_2, x_3, \ldots, x_n]^T \in \mathbb{X}$$

$$\partial_t u = F(t, x, u, \partial_x u, \partial_{xx} u, \ldots) \quad (t, x) \in [0, T] \times \mathbb{X}$$

$$B[u](t, x) = 0 \text{ for } (t, x) \in [0, T] \times \partial\mathbb{X}$$

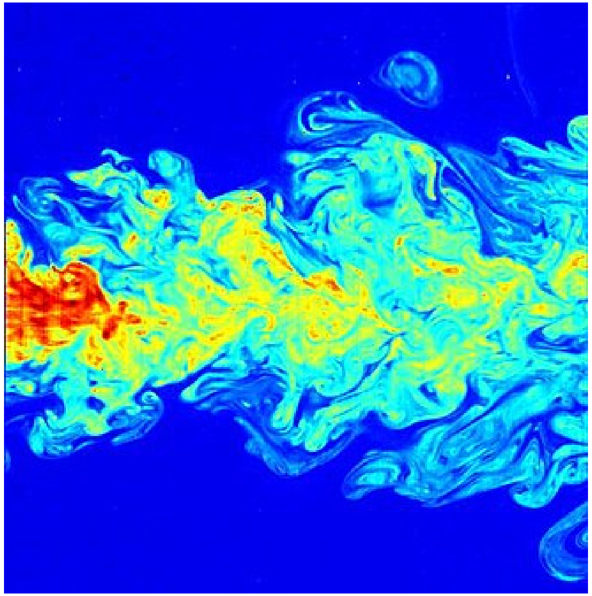
$$u(0, x) = u^0(x)$$

Conservation form

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{J}(\mathbf{u}) = 0$$

$$\frac{d\xi}{dt} + \boldsymbol{\nabla} \cdot \mathbf{f}(\xi) = 0$$

$$\frac{d}{dt} \int_V \xi \, dV = - \oint_{\partial V} \mathbf{f}(\xi) \cdot \boldsymbol{\nu} \, dS$$



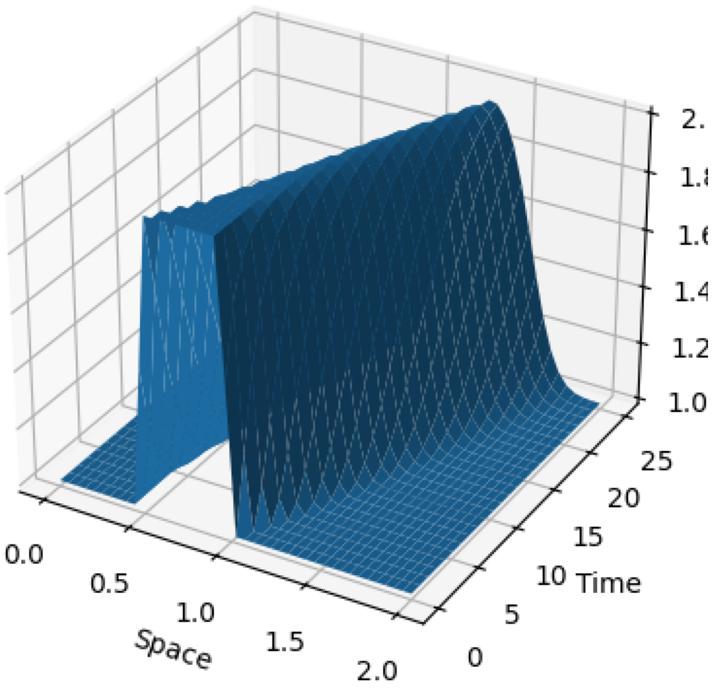
Finite difference example

wave.py

```python
import numpy
from matplotlib import pyplot as plt

# parameters
nx = 42
dx = 2 / (nx - 1)
nt = 25
dt = .025
c = 1

ut = numpy.ones((nt, nx))

# initial conditions
u = numpy.ones(nx)
u[10:20] = 2

# solve
for n in range(nt):
    old_u = u.copy()
    for i in range(1, nx):
        u[i] = old_u[i] - c * (dt / dx) * (old_u[i] - old_u[i - 1])
    ut[n] = u

# plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = numpy.linspace(start: 0, stop: 2, nx)
y = numpy.linspace(start: 0, nt, nt)
X, Y = numpy.meshgrid(*xi: x, y)
ax.plot_surface(X, Y, ut)
ax.set_xlabel('Space')
ax.set_ylabel('Time')

plt.show()
```

Plots

640x480 PNG (24-bit color) 78,77 k



Proudly stolen from CFDPython

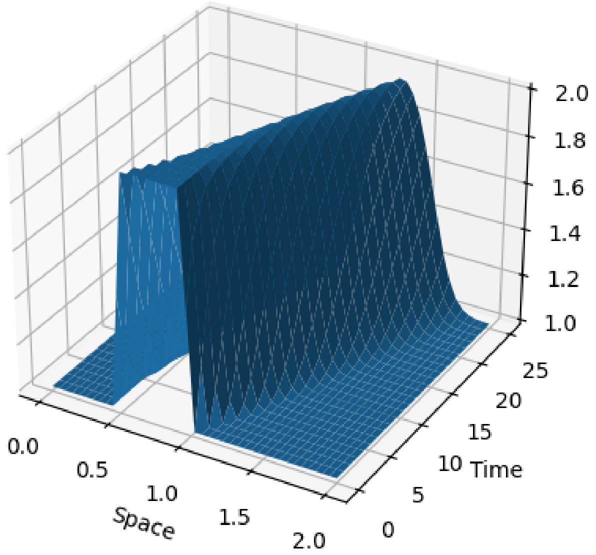
Finite difference example

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\delta x) - u(x)}{\delta x}$$

$$\frac{u_i^{n+1} - u_i^n}{\delta t} + c\frac{u_i^n - u_{i-1}^n}{\delta x} = 0$$

$$u_i^{n+1} = u_i^n - c\frac{\delta t}{\delta x}(u_i^n - u_{i-1}^n)$$



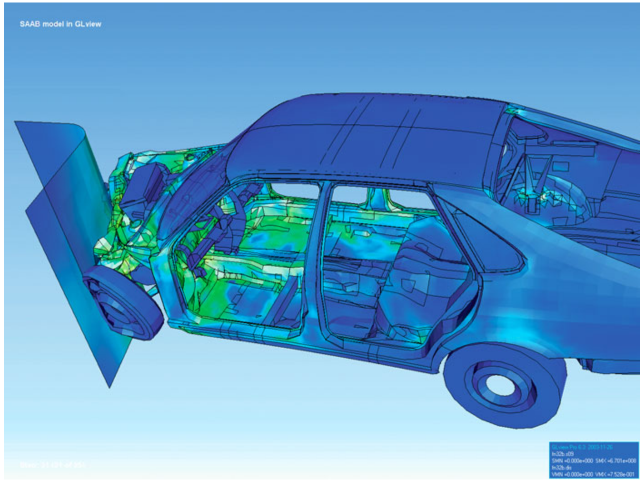
Proudly stolen from CFDPython

Numerical solvers

- Method of lines (MOL)
- Finite element method (FEM)
- PS
- etc.

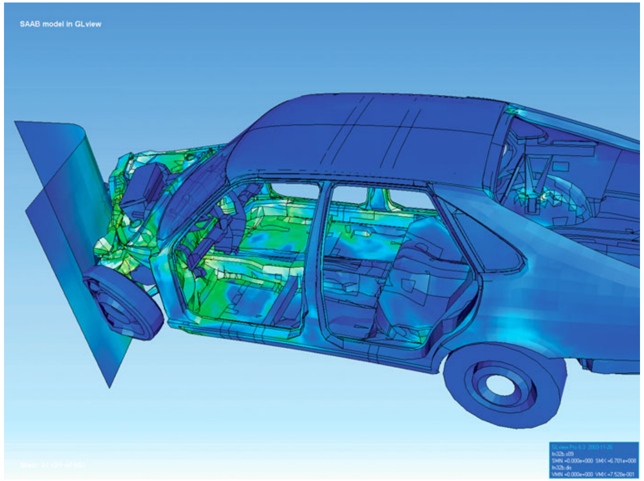
Numerical solvers

- Method of lines (MOL)
- Finite element method (FEM)
- PS
- etc.



Numerical solvers

- Method of lines (MOL)
- Finite element method (FEM)
- PS
- etc.



each with their different

→ Compute times

→ Accuracies / Errors

→ Sensitivities

→ Generalization abilities

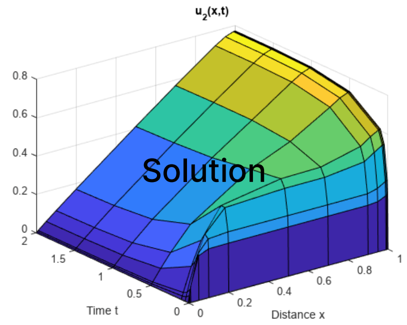
The plan

Numeric solver/
Analytical

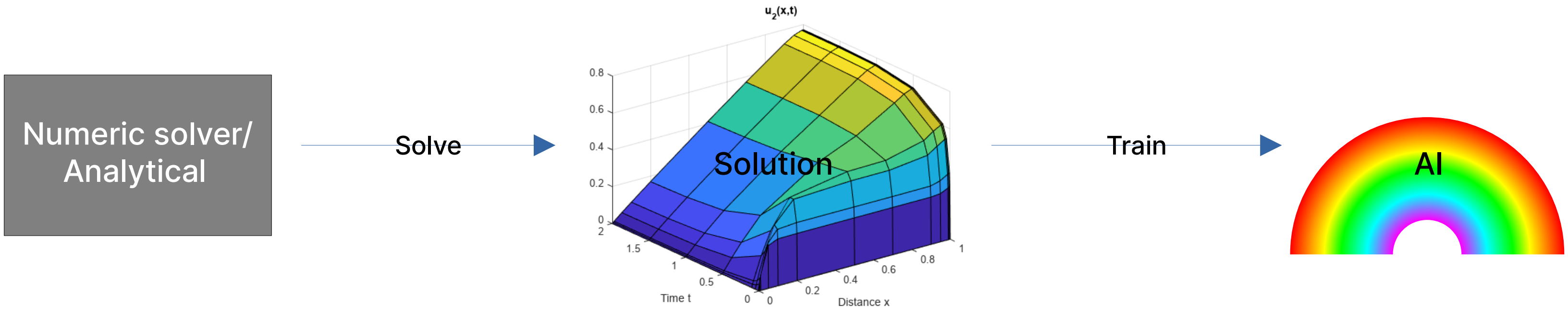
The plan

Numeric solver/ Analytical

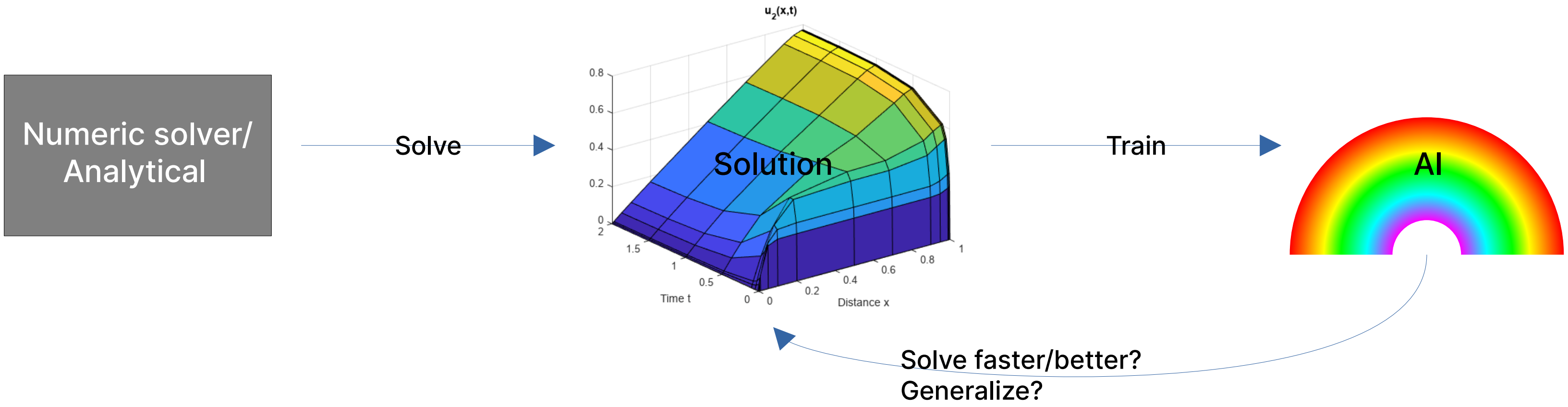
Solve



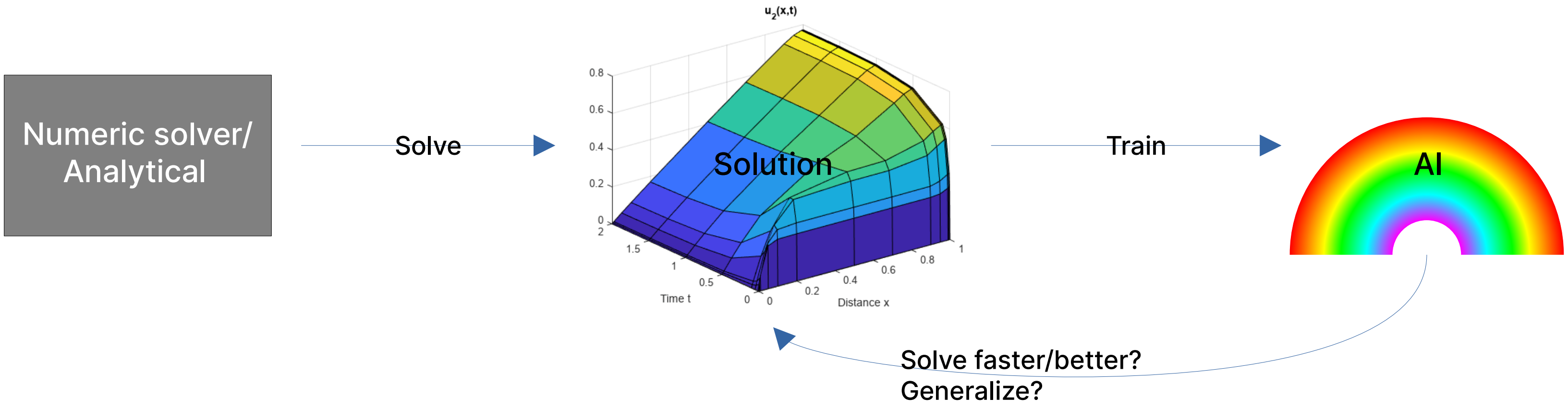
The plan


Numeric solver/
Analytical
Solve
$u_2(x,t)$
0.8
0.6
0.4
0.2
0
2
1.5
1
0.5
0
Time t
0
0.2
0.4
0.6
0.8
1
Distance x
Solution
Train
AI

The plan

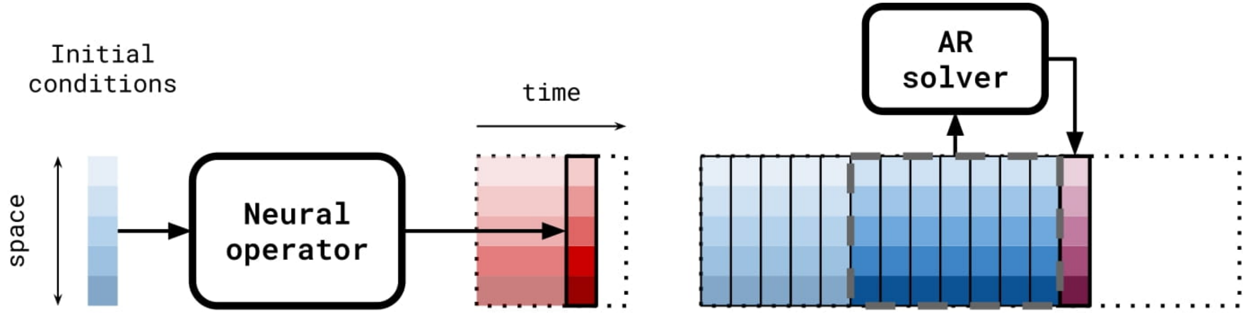

Numeric solver/
Analytical
Solve
u2(x,t)
Solution
Time t
Distance x
Train
AI
Solve faster/better?
Generalize?

The plan



Spoiler: This works quite good!

The options

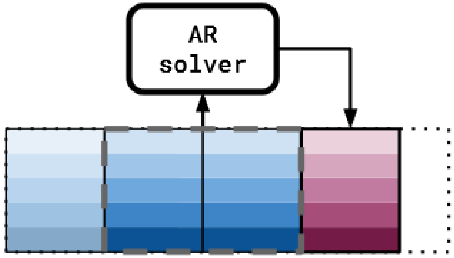


Neural operator
Mapping from initial
conditions to output

Autoregressive model
Mapping between temporally
consecutive time steps

$$\mathcal{M}(t, \mathbf{u}^0) = \mathbf{u}(t) \qquad \mathbf{u}(t + \Delta t) = \mathcal{A}(\Delta t, \mathbf{u}(t))$$

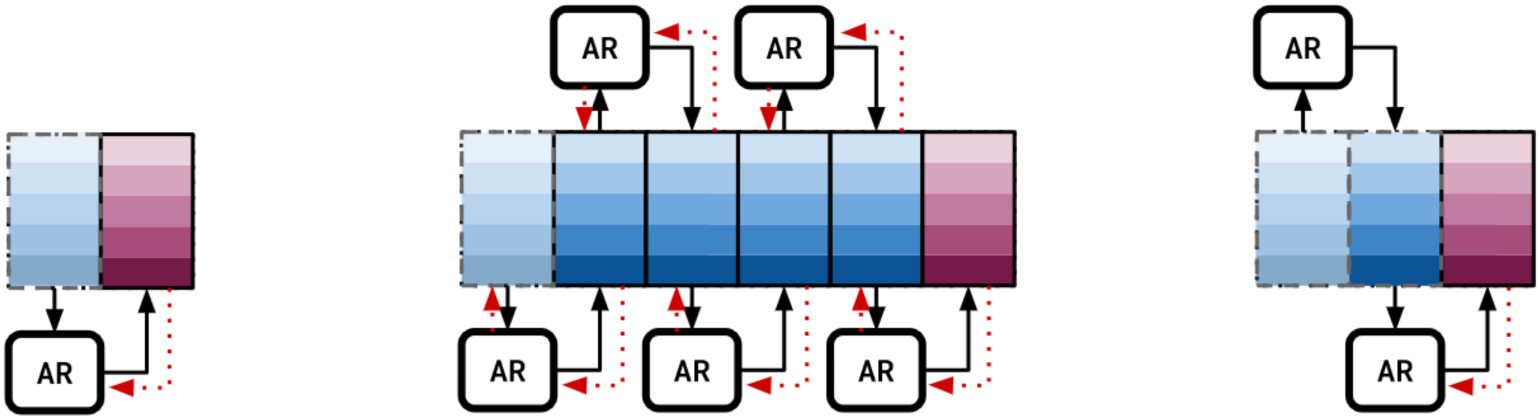
Temporal bundling



Temporal bundling
Fewer calls to solver reduces
error propagation speed

```python
class MP_PDE_Solver(torch.nn.Module):
    """
    MP-PDE solver class
    """
    def __init__(
        self,
        pde: PDE,
        time_window: int = 25,
        hidden_features: int = 128,
        hidden_layer: int = 6,
        eq_variables: dict = {}
    ):
```

Pushforward trick



One-step training
Gradients flow back one time step only

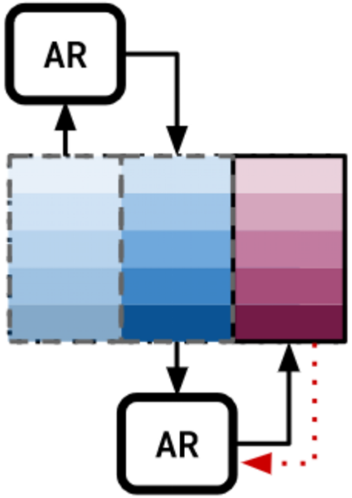
Unrolled training
Gradients flow back through all time steps

Pushforward training
Gradients flow only through last time step

Pushforward trick

https://www.youtube.com/watch?v=xDrArdzxJEI

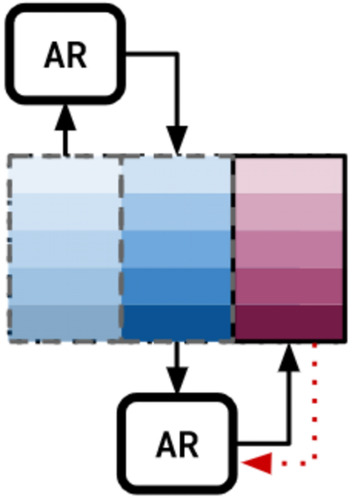
Pushforward trick



$$L_{\text{one-step}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k), \mathbf{u}^{k+1}) \right]$$

$$L_{\text{stability}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathbb{E}_{\boldsymbol{\epsilon}|\mathbf{u}^k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k + \epsilon), \mathbf{u}^{k+1}) \right] \right]$$

Pushforward trick

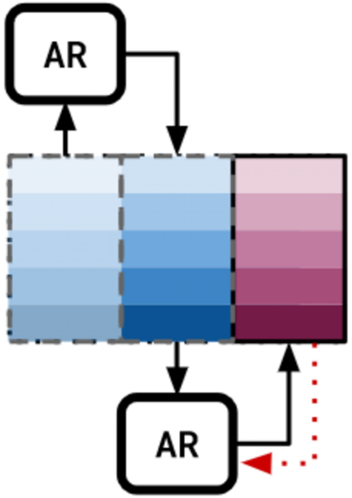


$$L_{\text{one-step}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k), \mathbf{u}^{k+1})\right]$$

$$L_{\text{stability}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathbb{E}_{\boldsymbol{\epsilon}|\mathbf{u}^k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k + \epsilon), \mathbf{u}^{k+1})\right]\right]$$

$$\epsilon \not\sim \mathcal{N}(\alpha, \beta^{-1})$$

Pushforward trick

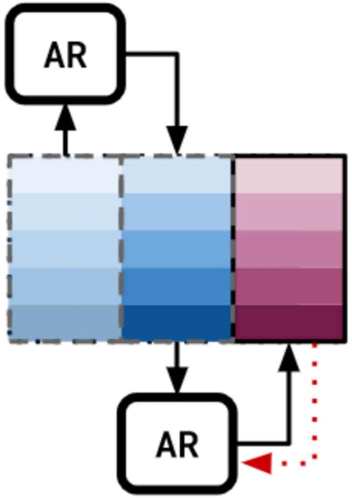


$$L_{\text{one-step}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k), \mathbf{u}^{k+1}) \right]$$

$$L_{\text{stability}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k, \mathbf{u}^k \sim p_k} \left[\mathbb{E}_{\boldsymbol{\epsilon}|\mathbf{u}^k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k + \boldsymbol{\epsilon}), \mathbf{u}^{k+1}) \right] \right]$$

$$\epsilon \not\sim \mathcal{N}(\alpha, \beta^{-1}) \qquad (\mathbf{u}^k + \boldsymbol{\epsilon}) = \mathcal{A}(\mathbf{u}^{k-1}) \text{ for } \mathbf{u}^{k-1}$$

Pushforward trick



$$L_{\text{one-step}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k,\mathbf{u}^k \sim p_k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k), \mathbf{u}^{k+1})\right]$$

$$L_{\text{stability}} = \mathbb{E}_k \mathbb{E}_{\mathbf{u}^{k+1}|\mathbf{u}^k,\mathbf{u}^k \sim p_k} \left[\mathbb{E}_{\boldsymbol{\epsilon}|\mathbf{u}^k} \left[\mathcal{L}(\mathcal{A}(\mathbf{u}^k + \boldsymbol{\epsilon}), \mathbf{u}^{k+1})\right]\right]$$

$$\epsilon \not\sim \mathcal{N}(\alpha, \beta^{-1}) \qquad (\mathbf{u}^k + \boldsymbol{\epsilon}) = \mathcal{A}(\mathbf{u}^{k-1}) \text{ for } \mathbf{u}^{k-1}$$

$$L_{\text{one-step}} + L_{\text{stability}}$$

Pushforward trick

```
# Unrolling of the equation which serves as input at the current step
# This is the pushforward trick!!!
with torch.no_grad():
    for _ in range(unrolled_graphs): # 0 or 1
        random_steps = [rs + graph_creator.tw for rs in random_steps]
        _, labels = graph_creator.create_data(u_super, random_steps)

        pred = model(graph)
        graph = graph_creator.create_next_graph(graph, pred, labels, random_steps).to(device)

pred = model(graph)
loss = criterion(pred, graph.y)
```

Pushforward half of the time?

```python
parser.add_argument('--unrolling', type=int,
                    default=1, help="Unrolling which proceeds with each epoch")
```

```python
# Sample number of unrolling steps during training (pushforward trick)
# Default is to unroll zero steps in the first epoch and then increase the max amount of unrolling st
max_unrolling = epoch if epoch <= args.unrolling else args.unrolling
unrolling = [r for r in range(max_unrolling + 1)]
```

```python
>>> [r for r in range(2)]
[0, 1]
```

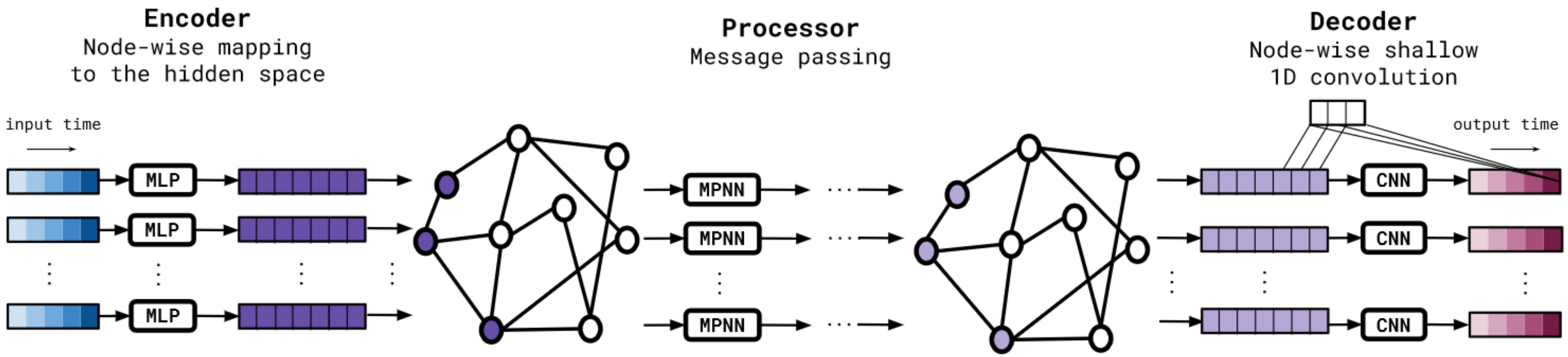
```python
for _ in range(unrolled_graphs):
```

Require: data, model, N, K, T
$t \leftarrow$ DrawRandomNumber $t \in \{1, ..., T\}$
input $\leftarrow$ data($t - K{:}t$)
for $n \in \{1, \ldots, N\}$ **do**
  input $\leftarrow$ model(input)
end for
target $\leftarrow$ data($t + NK{:}t + N(K + 1)$)
output $\leftarrow$ model(input)
loss $\leftarrow$ criterion(output, target)

PF in FNOs

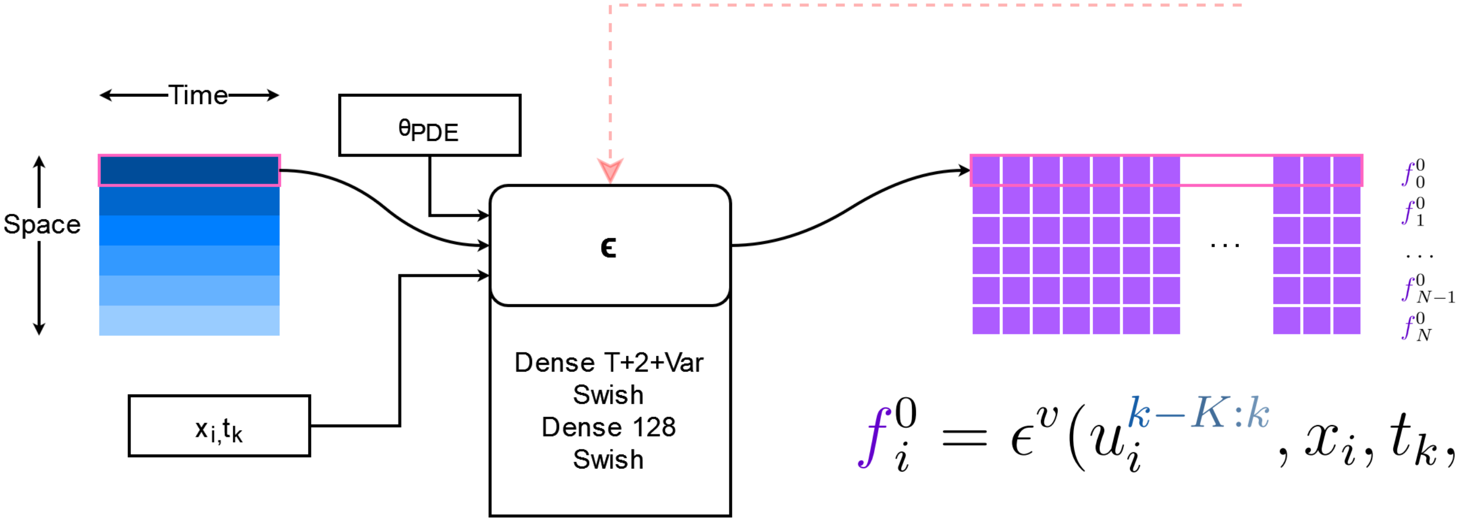
		Accumulated Error ↓					Runtime [s] ↓	
	(n_t, n_x)	WENO5	FNO-RNN	FNO-PF	MP-PDE-~~θ_{PDE}~~	MP-PDE	WENO5	MP-PDE
E1	(250, 100)	2.02	11.93	**0.54**				
E1	(250, 50)	6.23	29.98	**0.51**				
E1	(250, 40)	9.63	10.44	**0.57**				
E2	(250, 100)	1.19	17.09	2.53				
E2	(250, 50)	5.35	3.57	2.27				
E2	(250, 40)	8.05	3.26	2.38				
E3	(250, 100)	4.71	10.16	5.69				
E3	(250, 50)	11.71	14.49	5.39				
E3	(250, 40)	15.94	20.90	5.98				

The model



Encoder
Node-wise mapping
to the hidden space
Processor
Message passing
Decoder
Node-wise shallow
1D convolution
input time
output time
MLP
MLP
MLP
MPNN
MPNN
MPNN
CNN
CNN
CNN

Encoder

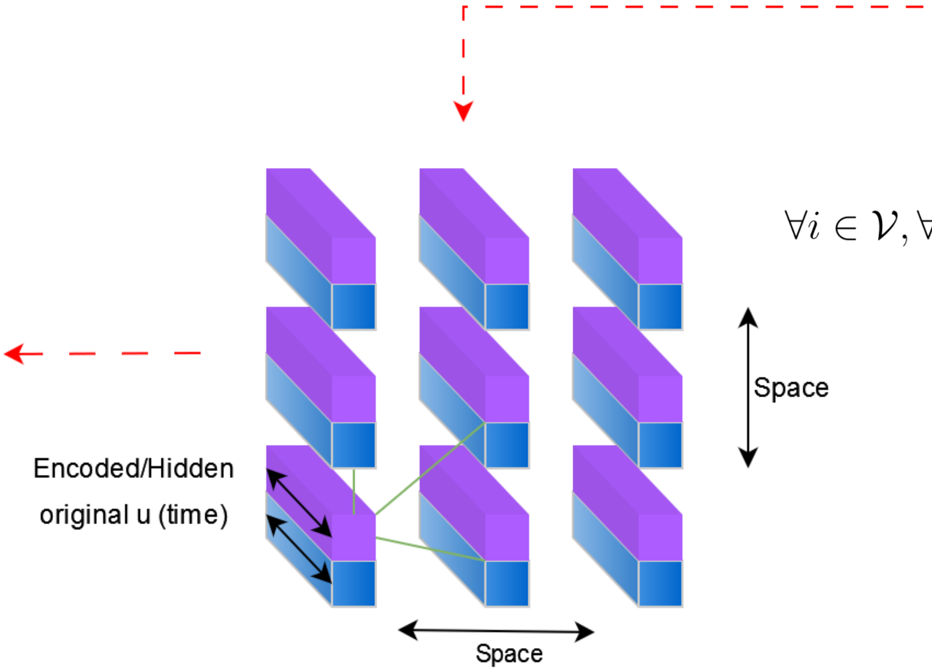


$$f_i^0 = \epsilon^v(u_i^{k-K:k}, x_i, t_k, \theta_{PDE})$$

```python
def forward(self, x):
    return x * torch.sigmoid(x)
```

$$\theta_{PDE} = (\alpha, \beta, \gamma, B.C.s, \dots)$$

Processor



$$\mathcal{N}(i) = \{j \in \mathcal{V} \,|\, dist(x_i, x_j) \leq r\}$$

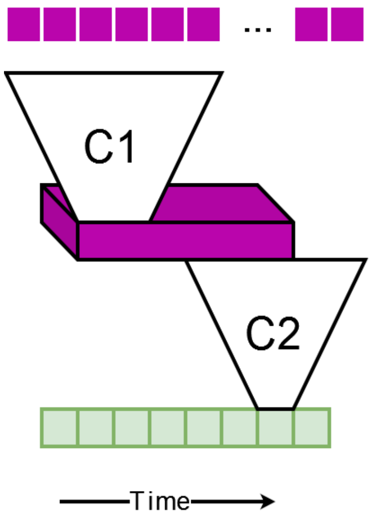
$$|\mathcal{N}(i)| \approx 6, m \in \{0,1,2,3,4,5,6\}$$

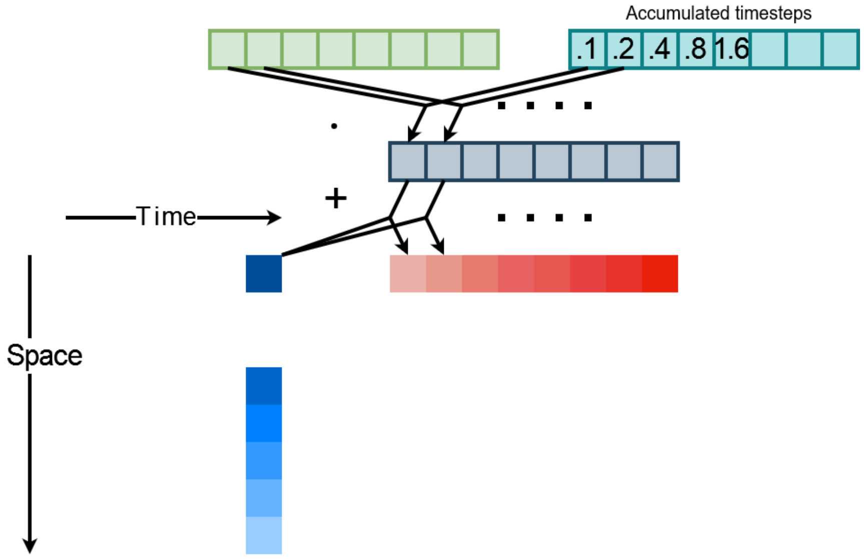
$$\forall i \in \mathcal{V}, \forall j \in \mathcal{N}(i): \quad m_{ij}^m = \phi_m(f_i^m, f_j^m, u_i^{k-K:k} - u_j^{k-K:k}, x_i - x_j, \theta_{PDE})$$

$$\forall i \in \mathcal{V}: \quad f_i^{m+1} = \psi_m(f_i^m, |\mathcal{N}(i)|^{-1} \sum_{j \in \mathcal{N}(i)} m_{ij}^m, \theta_{PDE})$$

ϕ	**ψ**
Dense 2f +t+d+Var Swish Dense 128 Swish	Dense 256+Var Swish Dense 128 Swish

Decoder





C1

1D Convolution
1 to 8 channels
Kernel: 16
Stride: 3

C2

1D Convolution
8 to 1 channel
Kernel: 14
Stride: 1

```
# Decoder (formula 10 in the paper)
dt = (torch.ones(1, self.time_window) * self.pde.dt).to(h.device)
dt = torch.cumulativeSum(dt, dim=1)
# [batch*n_nodes, hidden_dim] -> 1DCNN([batch*n_nodes, 1, hidden_dim]) -> [batch*n_nodes, time_window]
diff = self.output_mlp(h[:, None]).squeeze(1)
out = u[:, -1].repeat(self.time_window, 1).transpose(0, 1) + dt * diff

return out
```

Backpropagation?

Backpropagation?

Backpropagation?

PyTorch
Autograd

```
pred = model(data)
loss = criterion(pred, labels)
loss = torch.sqrt(loss)
loss.backward()
optimizer.step()
```

Wave results

$$\partial_{tt} u - c^2 \partial_{xx} u = 0, \qquad x \in [-8, 8]$$

Wave results

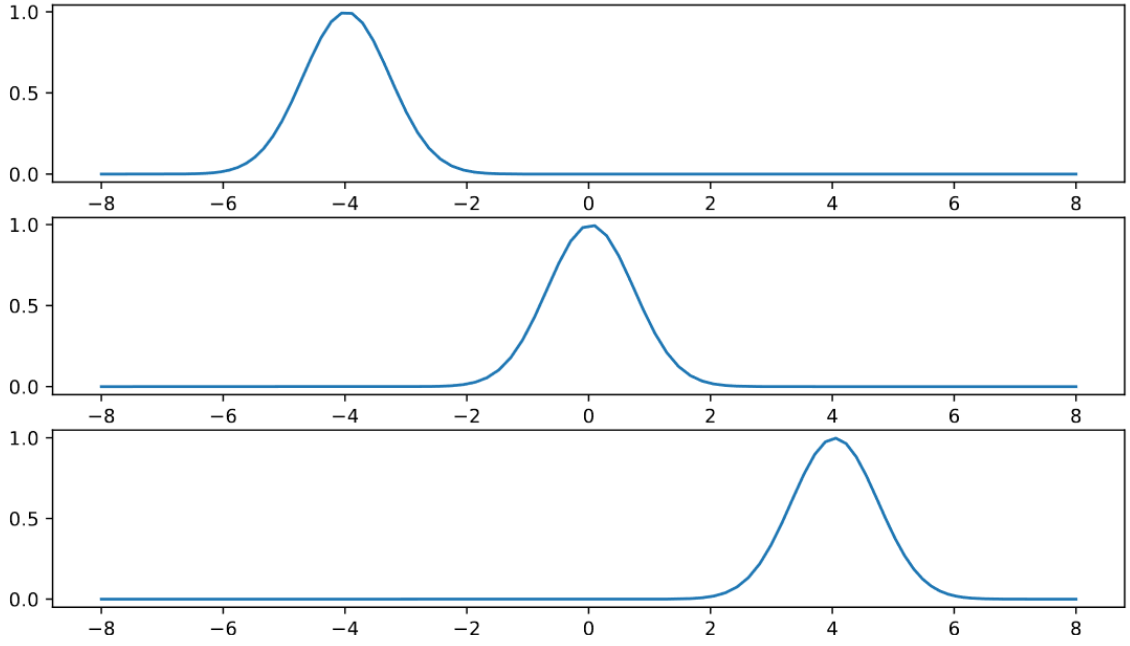
$$\partial_{tt} u - c^2 \partial_{xx} u = 0, \qquad x \in [-8, 8]$$

$$\partial_t [u, v] - [v, c^2 \partial_{xx} u] = 0$$

Wave results

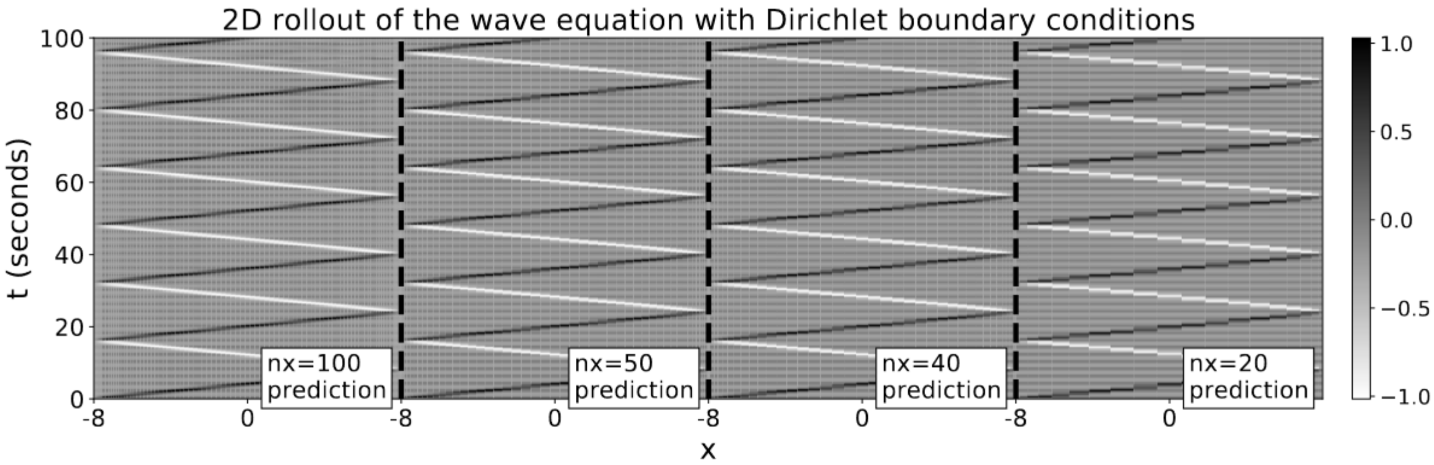
$$\partial_{tt} u - c^2 \partial_{xx} u = 0, \qquad x \in [-8, 8]$$

$$\partial_t [u, v] - [v, c^2 \partial_{xx} u] = 0$$

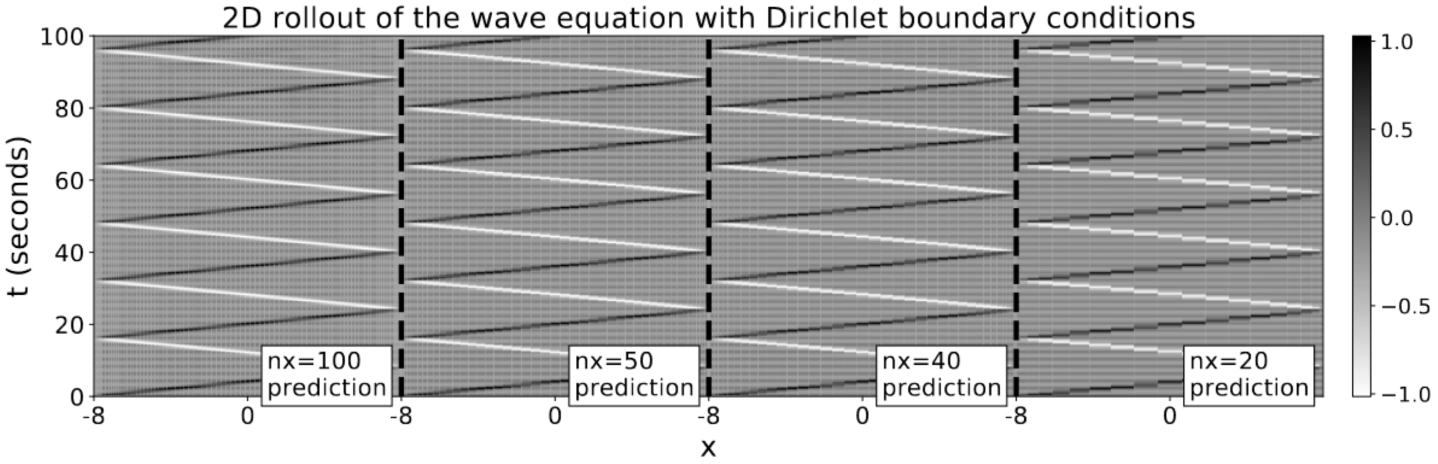


Wave results

$$\partial_{tt} u - c^2 \partial_{xx} u = 0, \qquad x \in [-8, 8]$$
$$\partial_t [u, v] - [v, c^2 \partial_{xx} u] = 0$$



Wave results

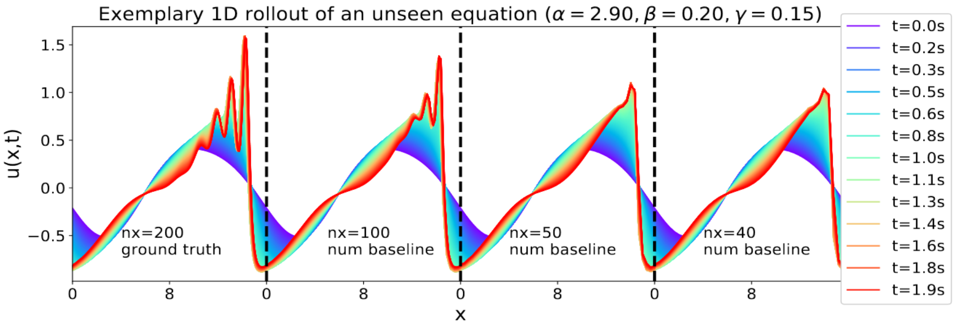


	$\frac{1}{n_x}\sum_{x,t}$ MSE ↓ (**WE1**)		$\frac{1}{n_x}\sum_{x,t}$ MSE ↓ (**WE2**)		$\frac{1}{n_x}\sum_{x,t}$ MSE ↓ (**WE3**)			Runtime [s] ↓	
(n_t, n_x)	PS	MP-PDE	PS	MP-PDE	PS	MP-PDE ~~θ_{PDE}~~	MP-PDE	PS	MP-PDE
(250, 100)	0.004	0.137	0.004	0.111	0.004	38.775	0.097	0.60	0.09
(250, 50)	0.450	0.035	0.681	0.034	0.610	20.445	0.106	0.35	0.09
(250, 40)	194.622	0.042	217.300	0.003	204.298	16.859	0.219	0.25	0.09
(250, 20)	breaks	0.059	breaks	0.007	breaks	17.591	0.379	0.20	0.07

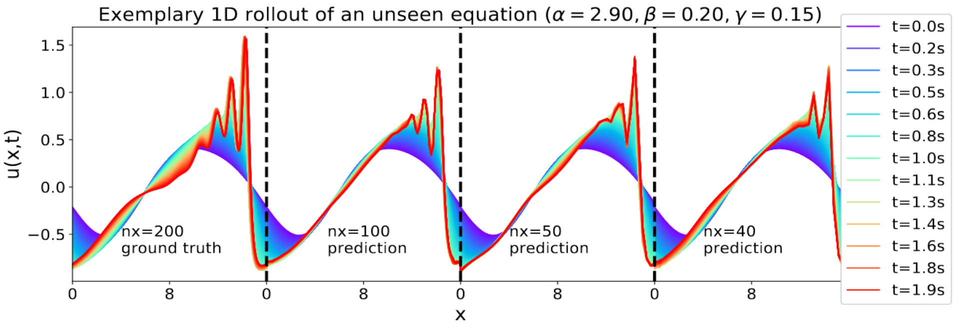
Results

$$[\partial_t u + \partial_x(\alpha u^2 - \beta \partial_x u + \gamma \partial_{xx} u)](t,x) = \delta(t,x)$$

Numeric solver



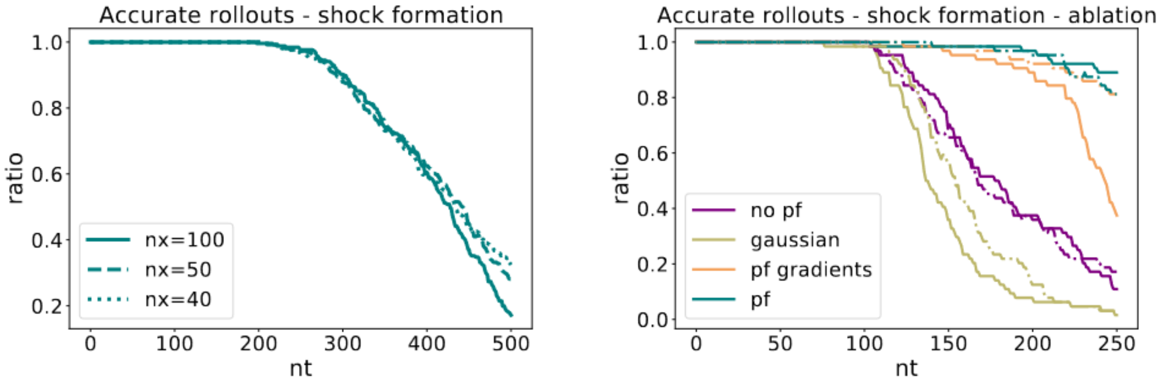
Model



$$\theta_{PDE} = (\alpha, \beta, \gamma, B.C.s, \dots)$$

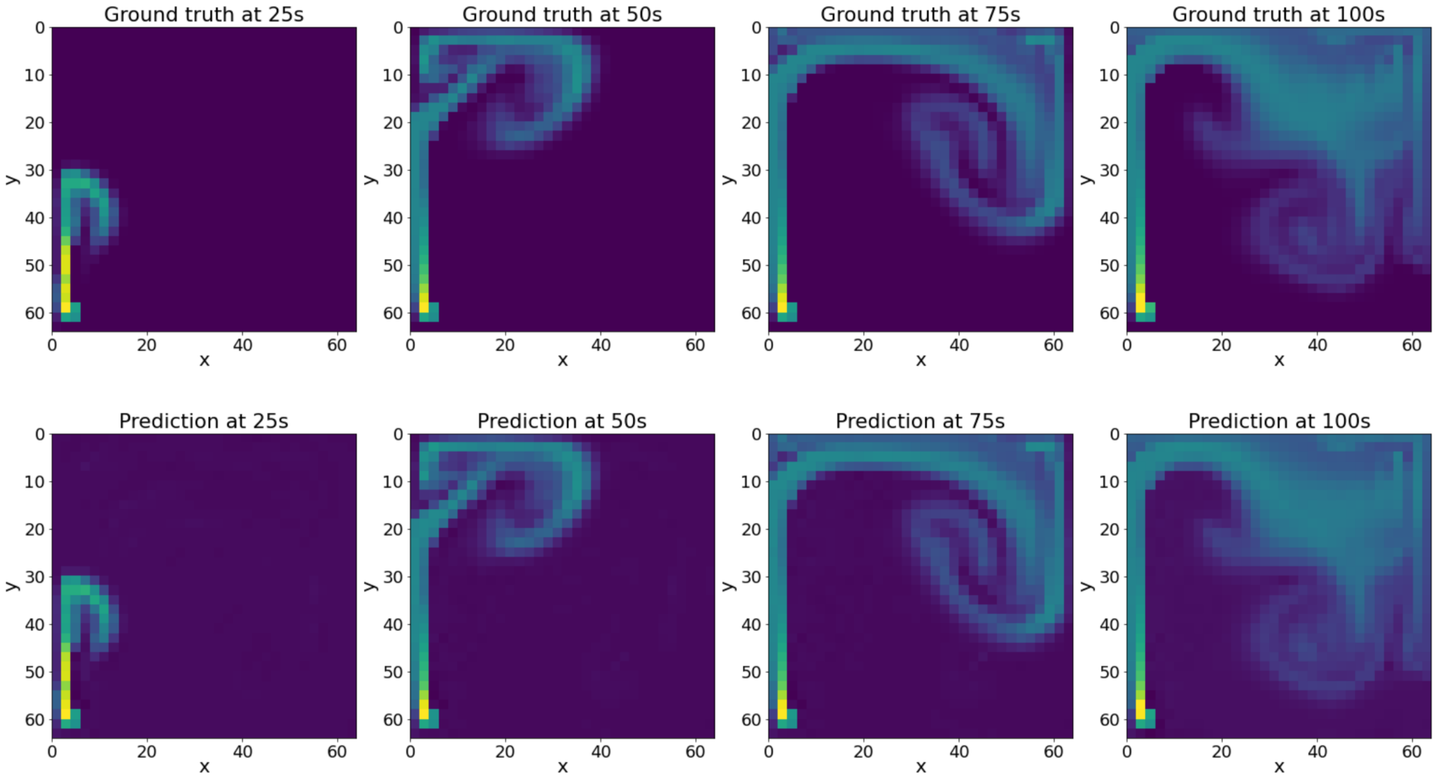
Results

		Accumulated Error ↓					Runtime [s] ↓	
	(n_t, n_x)	WENO5	FNO-RNN	FNO-PF	MP-PDE-~~θ_{PDE}~~	MP-PDE	WENO5	MP-PDE
E1	(250, 100)	2.02	11.93	**0.54**	-	1.55	1.9	0.09
E1	(250, 50)	6.23	29.98	**0.51**	-	1.67	1.8	0.08
E1	(250, 40)	9.63	10.44	**0.57**	-	1.47	1.7	0.08
E2	(250, 100)	1.19	17.09	2.53	1.62	**1.58**	1.9	0.09
E2	(250, 50)	5.35	3.57	2.27	1.71	**1.63**	1.8	0.09
E2	(250, 40)	8.05	3.26	2.38	1.49	**1.45**	1.7	0.08
E3	(250, 100)	4.71	10.16	5.69	4.71	**4.26**	4.8	0.09
E3	(250, 50)	11.71	14.49	5.39	10.90	**3.74**	4.5	0.09
E3	(250, 40)	15.94	20.90	5.98	7.78	**3.70**	4.4	0.09

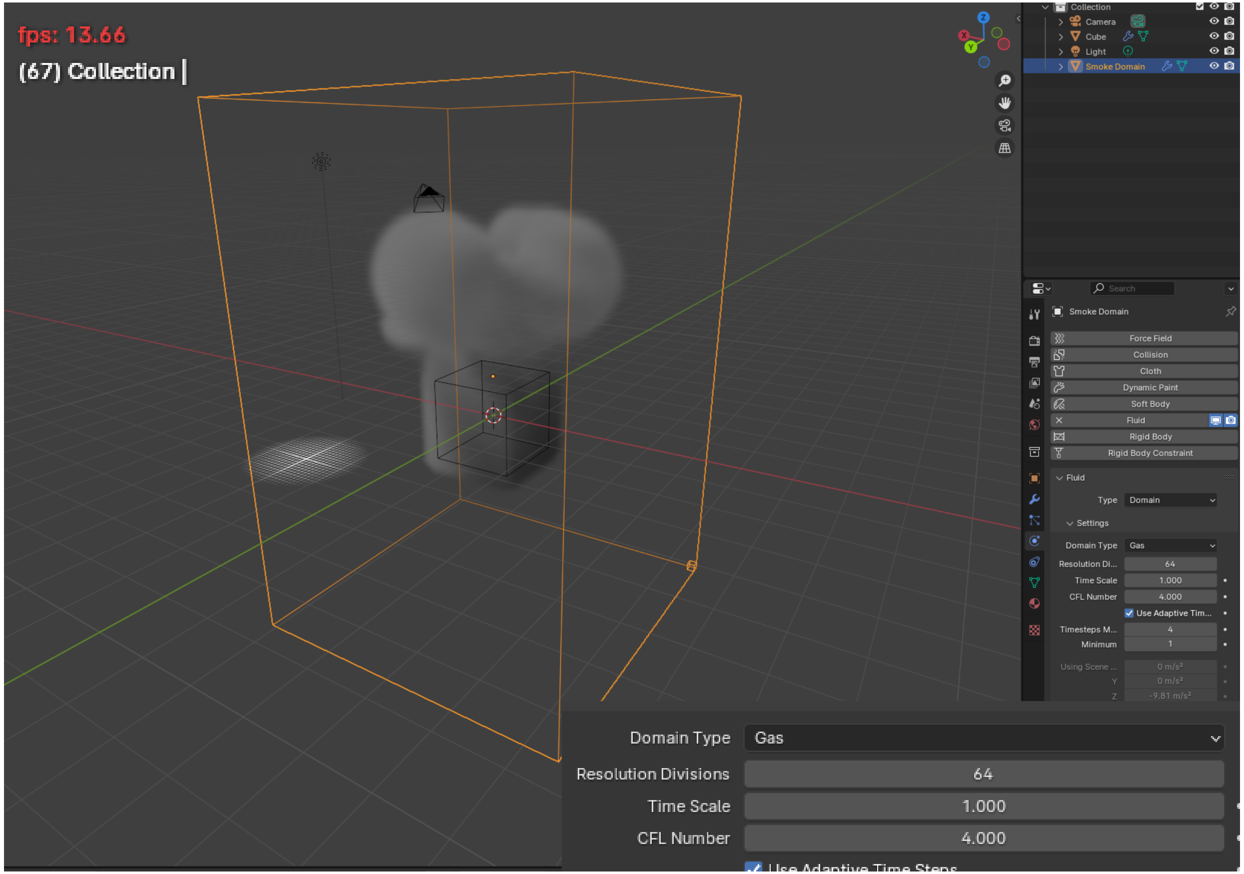


(a) (b)

Results


Ground truth at 25s
Ground truth at 50s
Ground truth at 75s
Ground truth at 100s
Prediction at 25s
Prediction at 50s
Prediction at 75s
Prediction at 100s
x
y
0
10
20
30
40
50
60

Impact



fps: 13.66
(67) Collection
Collection
Camera
Cube
Light
Smoke Domain
Search
Smoke Domain
Force Field
Collision
Cloth
Dynamic Paint
Soft Body
Fluid
Rigid Body
Rigid Body Constraint
Fluid
Type Domain
Settings
Domain Type Gas
Resolution Di... 64
Time Scale 1.000
CFL Number 4.000
Use Adaptive Tim...
Timesteps M... 4
Minimum 1
Domain Type Gas
Resolution Divisions 64
Time Scale 1.000
CFL Number 4.000

CT

Takeaways

- AI will take over the world

Takeaways

- AI will ~~take over~~ at least revolutionize the world

Takeaways

- AI will ~~take over~~ at least revolutionize the world
- Creativity in design is vital for successful models

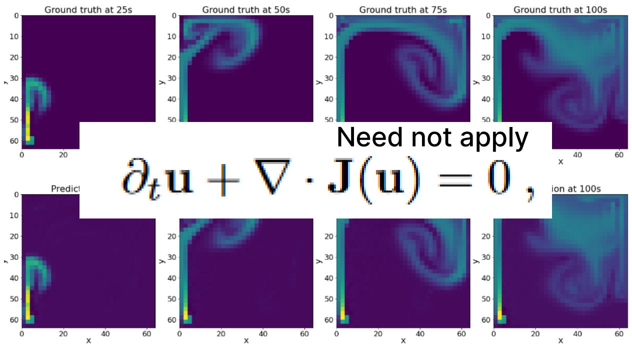
Takeaways

- AI will ~~take over~~ at least revolutionize the world
- Creativity in design is vital for successful models
- Exploring new AI-driven approaches is fun!

Note to the authors

survival times using pushforward (no pf), n

```
losses_ps = ['breaks', '194.622', '0.450', '0.004']
losses_mp = ['0.059', '0.042', '0.035', '0.137']

runtimes_ps = ['0.20', '0.25', '0.35', '0.60']
runtimes_mp = ['0.07', '0.09', '0.09', '0.09']
```



Need not apply

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{J}(\mathbf{u}) = 0\,,$$

$$\psi\left(\mathbf{f}_i^m,\ \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}^m,\ \boldsymbol{\theta}_{\text{PDE}}\right)$$

?

```
aggr='mean')
```

???

```
>>> [r for r in range(2)]
[0, 1]
```

Require: data, model, N, K, T
$t \leftarrow$ DrawRandomNumber $t \in \{1, ..., T\}$
input $\leftarrow$ data($t - K$:t)
for $n \in \{1, \ldots, N\}$ **do**
input $\leftarrow$ model(input)
end for
target $\leftarrow$ data($t + NK$:$t + N(K+1)$)
output $\leftarrow$ model(input)
loss $\leftarrow$ criterion(output, target)

Thanks!