MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Data-driven Parameter Discovery of One-dimensional Burgers' Equation Using Physics-Informed Neural Network

Eduardo Furlan Miranda
Stephan Stephany
Roberto Pinto Souto

Qualification Exam - Doctorate CAP/INPE

Table of Contents

Why PINN ?

- Physics-Informed Neural Network (PINN)
    - Data-driven model derived using law of physics described by general nonlinear PDEs and constraints (ICs and BCs)
- PINN performs PDE parameter discovery (inverse problem) or PDE solution (direct problem)
- Suitable for sparse, limited, incomplete, or noisy data, irregular domains and complex non-linear patterns
- No solver or discretization/grid required

Why PINN ?

- Recent approach for identifying and solving dynamical systems involving PDEs
- The literature show promising speedups by porting a standard module to PINN in a weather numerical model
- Availability of frameworks like NVIDIA Modulus and Uber Horovod for GPU execution

1 Introduction

Objectives

- Comparing accuracy of PINN and a numerical method (SINDy) for PDE parameter discovery of a toy problem
- Evaluate PINN performance for different CP sizes and hyperparameters
- The PINN is trained resulting in a model then used for solution (parameter discovery & solution)
- Visual evaluation of the PINN solution using the exact solution as reference

Related work

- Speedup of 3 by refactoring the solver and replacing the gas optics module with a PINN (Ukkonen et a., 2020)
- Speedup of 7 in the ECMWF Long-wave Radiative Transfer model (Chevallier et al., 2020)
- 10 to 10^5 times acceleration of the Longwave Radiation parameterization for the NCAR CAM and NSIPP GCM (Krasnopolsky et al., 2006)

Challenges

- Lack of public documentation for PINN downloadable implementations
- New PINN implementations for real-world applications
- Porting parts/modules of existing code to PINNs
- PINN execution on GPUs, using new frameworks like Modulus

Two parts in this work

1. Comparison of accuracy and processing time of PINN and a standard numerical method (SINDy) in the inverse and direct problem (toy problem) - execution on a local PC

2. Analysis of size of the CP set and hyperparameters in the PINN performance - execution on the LNCC SDumont supercomputer

2 Approaches

PINN - some approaches and resources

- MLP: the most common architecture

- Other architectures
  - Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Auto-Encoder (AE), Deep Belief Network (DBN), Generative Adversarial Network (GAN) and Bayesian Deep Learning (BDL)

- PINN variations
  - Variational hp-VPINN, conservative PINN (CPINN), and physically constrained DNNs (PCNN), Conservative PINN (CPINN), Conservative PINN (CPINN)

PINN - some approaches and resources

- Current research on PINNs explores architectures, activation functions, loss functions, and gradient optimization techniques
- The PINN mainstream is still the PDE direct problem
- The number of works PINNs to solve PDE inverse problems has been increasing

PINN - some approaches and resources

- PINNs may model the PDE with unknown IC and BC (called soft BC)

- PCNNs, a class of *data-free* PINNs, impose known IC and BC (hard BC) via a customized DNN architecture, that also include the PDE in the loss function

- Frameworks and implementations
    - Deep Ritz Method (DRM), Deep Ritz Method (DRM), Deep Galerkin Method (DGM), hp-VPINN

3 Toy Problem

Toy problem - 1D Burgers' equation

- Velocity field *u* of a fluid (dimension *x* and time *t*)

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \qquad x \in [-1, 1],\ t \in [0, 1],$$

IC: $u(0, x) = -\sin(\pi x),$

BC: $u(t, -1) = u(t, 1) = 0$

- Unknown parameters are the coefficients of the differential operators: *λ_1 = 1.0,* *λ_2 = ν = 0.01/π or 0.1/π (smallest viscosity causes discontinuities)*

PINN parameter discovery

- MLP architecture: 2-neuron input layer, 1-to-8 hidden layers using hyperbolic tangent as activation function, 10-to-30 neurons per hidden layer and a single-neuron output layer

- The loss function uses the Mean Squared Error (MSE), being minimized by the Generalized Limited-memory Broyden-Fletcher-Goldfarb- Shanno (L-BFGS) optimization algorithm

- Each iteration equals one epoch (singe batch per epoch)

PINN training input data (set of CPs)

- Problem sizes (1D grid points x timesteps)
    - 128x64, 256x100, 256x128, 512x256
    - Exact field given by Gaussian Quadrature Method (GQM)

- No division of PINN input data into training, validation and test sets (training -> resulting model)
    - Random sample of 2,000 CPs from given dataset
    - PDE parameters discovery (from training)
    - PDE predicted solution (no further training)

PINN loss function

- Two-term MSE (iterations k and k-1)

$$MSE^k = MSE_u^k + MSE_f^{k-1}$$

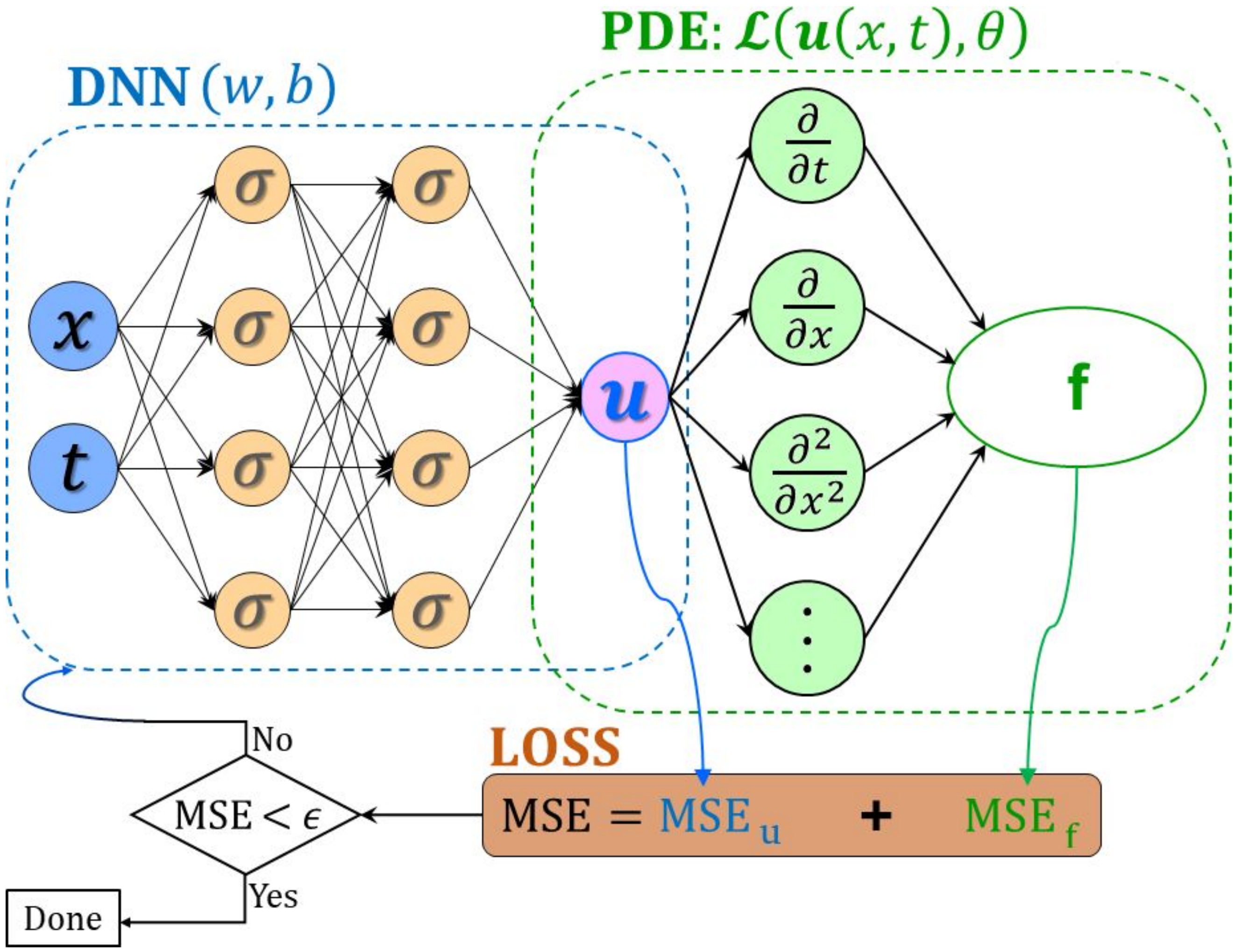
- MSE_u - PINN solution matching of the exact data points for the set of CPs at iteration k

$$MSE_u^k = \frac{1}{N}\sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2$$

- MSE_f - residual of the known PDE for the set of CPs predicted by the PINN at iteration (k-1)

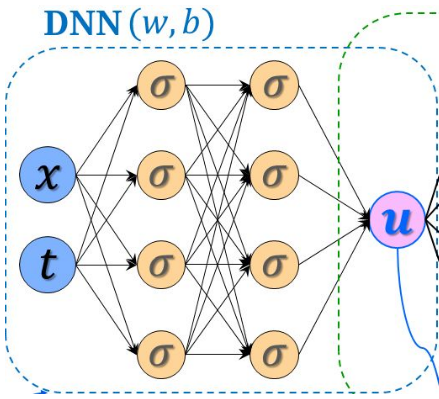
$$MSE_f^{k-1} = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2$$

PINN loss function



PDE: $\mathcal{L}(u(x,t),\theta)$
DNN (w,b)
x
t
σ
u
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\vdots$
f
LOSS
$\mathrm{MSE} = \mathrm{MSE}_u + \mathrm{MSE}_f$
$\mathrm{MSE} < \epsilon$
No
Yes
Done

PINN implementation: u(t, x)



Predicted solution using the DNN

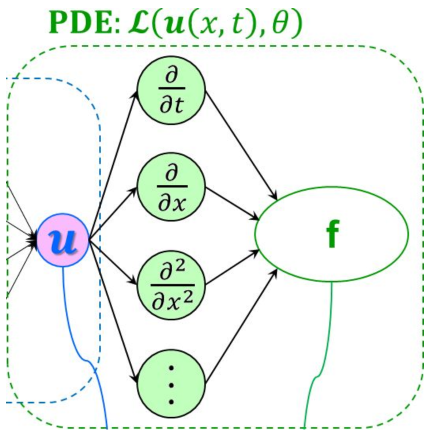
TensorFlow 1.15 (GPU)

```
def neural_net(X, weights, biases):
    num_layers = len(weights) + 1
    H = 2.0 * (X - lb) / (ub - lb) - 1.0
    for l in range(0, num_layers - 2):
        W = weights[l]
        b = biases[l]
        H = tf.tanh(tf.add(tf.matmul(H, W), b))
    W = weights[-1]
    b = biases[-1]
    Y = tf.add(tf.matmul(H, W), b)
    return Y

def net_u(t, x):
    u = neural_net(tf.concat([x, t], 1), weights, biases)
    return u
```

PINN implementation: f(t, x)

`Predicted solution using the PDE`


PDE: $\mathcal{L}(\boldsymbol{u}(x,t),\theta)$
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\vdots$
u
f

```
def net_f(x, t):
    lambda_1 = lambda_1
    lambda_2 = tf.exp(lambda_2)
    u = net_u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
    return f
```

TensorFlow 1.15 (GPU)

`tf.gradients` is used in the gradient-based training and optimization algorithm, and uses automatic differentiation to compute gradients (derivatives)

PINN implementation: loss function

LOSS

$$MSE = MSE_u + MSE_f$$

```
u_tf = u  # exact solution

u_pred = net_u(t, x)  # predicted solution using the DNN

f_pred = net_f(t, x)  # predicted solution using the PDE


loss = (tf.reduce_mean(tf.square(u_tf - u_pred)) +

             tf.reduce_mean(tf.square(f_pred)))
```

SINDy parameter discovery

- Sparse Identification of Nonlinear Dynamical Systems (SINDy) method (Brunton et al., 2016)
- Uses sparse regression to create a linear combination of basis functions to capture the dynamic behavior of the considered physical system
- Iterative optimization using an objective function
- Applications: linear and nonlinear oscillators, chaotic systems, fluid dynamics, and others

SINDy

- The temporal evolution of *x(t)* is modeled using the nonlinear function

$$\frac{d}{dt}x(t) = f(x(t))$$

- The vector $x(t)=[x_1(t), x_2(t), \dots x_□(t)]^T$ represents the state of the physical system at time *t*

SINDy

- The problem solved by SINDy is

$$\dot{X} = \Theta(X)\,\Xi$$

- *Θ* : matrix *f(x(t))* of basis functions applied to the input data *X*, i.e. *Θ(X)*

- *Ξ* : matrix of coefficients that indicates which terms in Θ (X) are significant
  - reconstructs the governing equations of the dynamical system

- Along the iterations, these coefficients are optimized until achieving convergence

PySINDy implementation of SINDy

Includes 4 different Sparse Regression Optimizers

- Sequentially Threshold Least Squares (STLSQ)
- Orthogonal Least Squares of Forward Regression (FROLS)
- Sparse Relaxed Regularized Regression (SR3)
- Sparse Stepwise Regression (SSR)

http://pysindy.readthedocs.io

Python code snippet that implements SINDy

```
optimizer = ps.STLSQ(threshold=2, alpha=1e-5,
                     normalize_columns=True)
model = ps.SINDy(feature_library=pde_lib,
                 optimizer=optimizer,
                 feature_names=["u"])
model.fit(u, t=dt)
model.print()
```

Computing environment

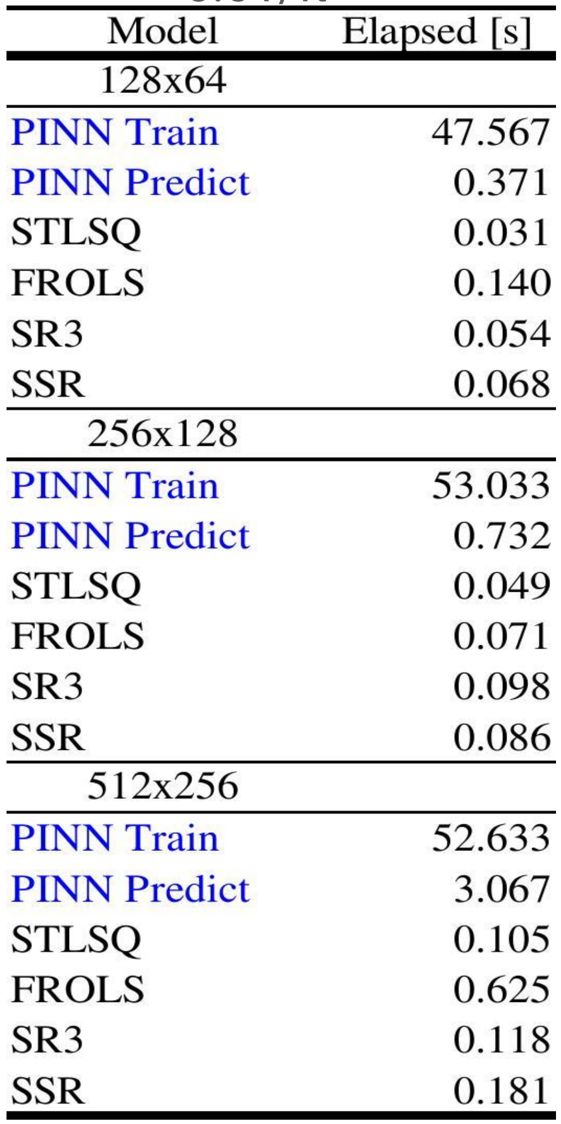
- PC local machine 6-core Intel i7 9750h CPU, 8 GB of main memory, and an NVIDIA GTX 1050 GPU (768 CUDA cores and 3 GB of memory)
- LNCC SDumont single Bull Sequana X1120 processing node with two 24-core Intel Xeon Gold 6252 Skylake 2.1 GHz processors (total of 48 CPU cores), 384 GB of main memory and 4 Nvidia Volta V100 GPUs
- Python 3.7, TensorFlow 1.15, PySINDy 1.7.5

4 Results

PINN and SINDy models - elapsed time [s]

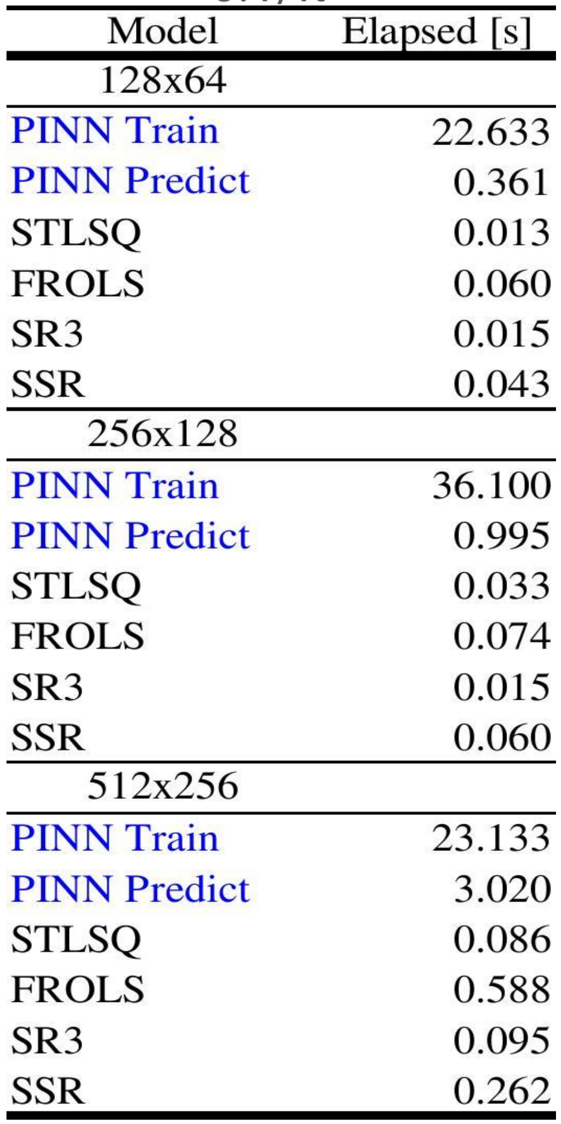
(local PC machine)

0.01/π



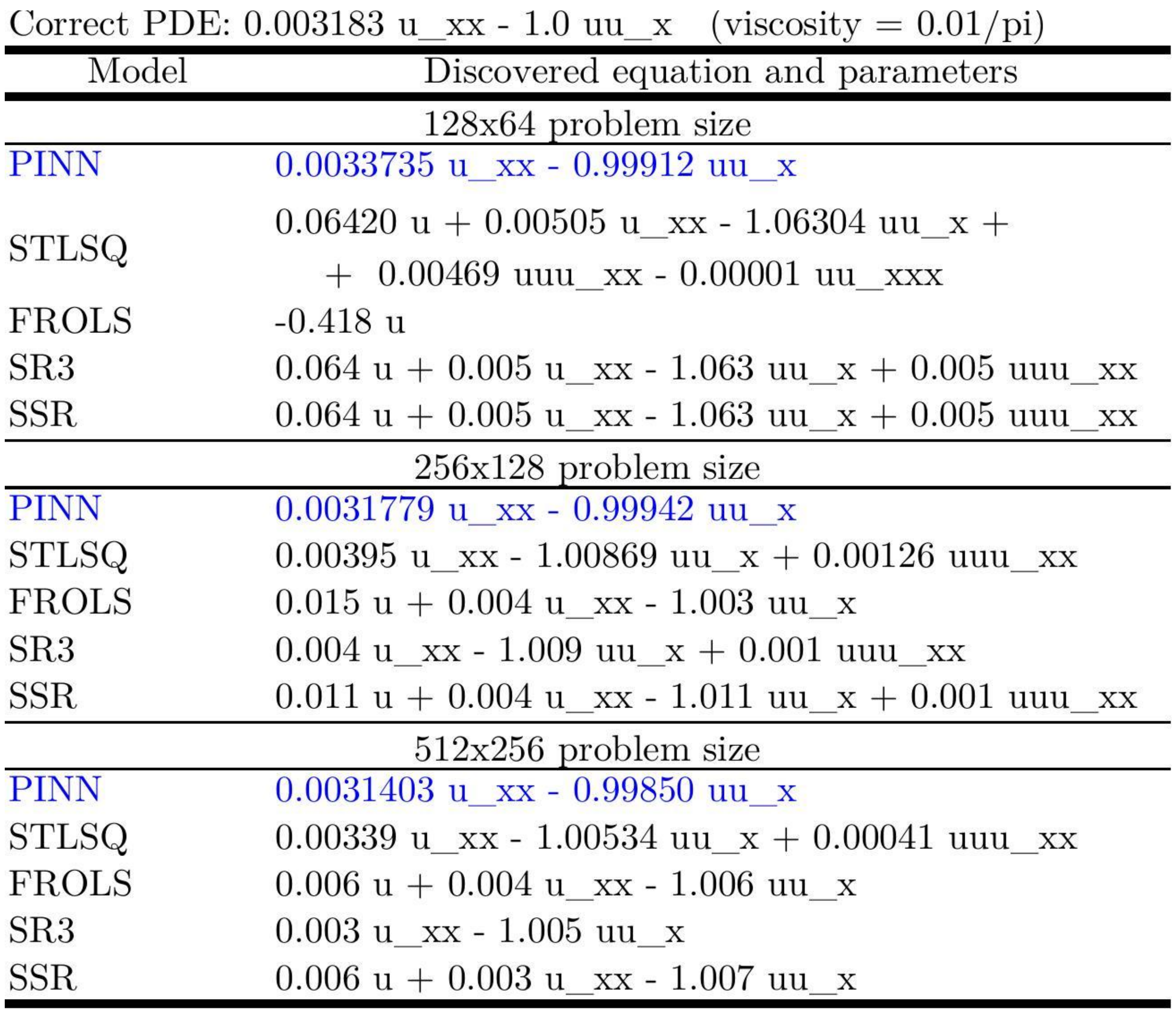
Model	Elapsed [s]
128x64	
PINN Train	47.567
PINN Predict	0.371
STLSQ	0.031
FROLS	0.140
SR3	0.054
SSR	0.068
256x128	
PINN Train	53.033
PINN Predict	0.732
STLSQ	0.049
FROLS	0.071
SR3	0.098
SSR	0.086
512x256	
PINN Train	52.633
PINN Predict	3.067
STLSQ	0.105
FROLS	0.625
SR3	0.118
SSR	0.181

0.1/π



Model	Elapsed [s]
128x64	
PINN Train	22.633
PINN Predict	0.361
STLSQ	0.013
FROLS	0.060
SR3	0.015
SSR	0.043
256x128	
PINN Train	36.100
PINN Predict	0.995
STLSQ	0.033
FROLS	0.074
SR3	0.015
SSR	0.060
512x256	
PINN Train	23.133
PINN Predict	3.020
STLSQ	0.086
FROLS	0.588
SR3	0.095
SSR	0.262

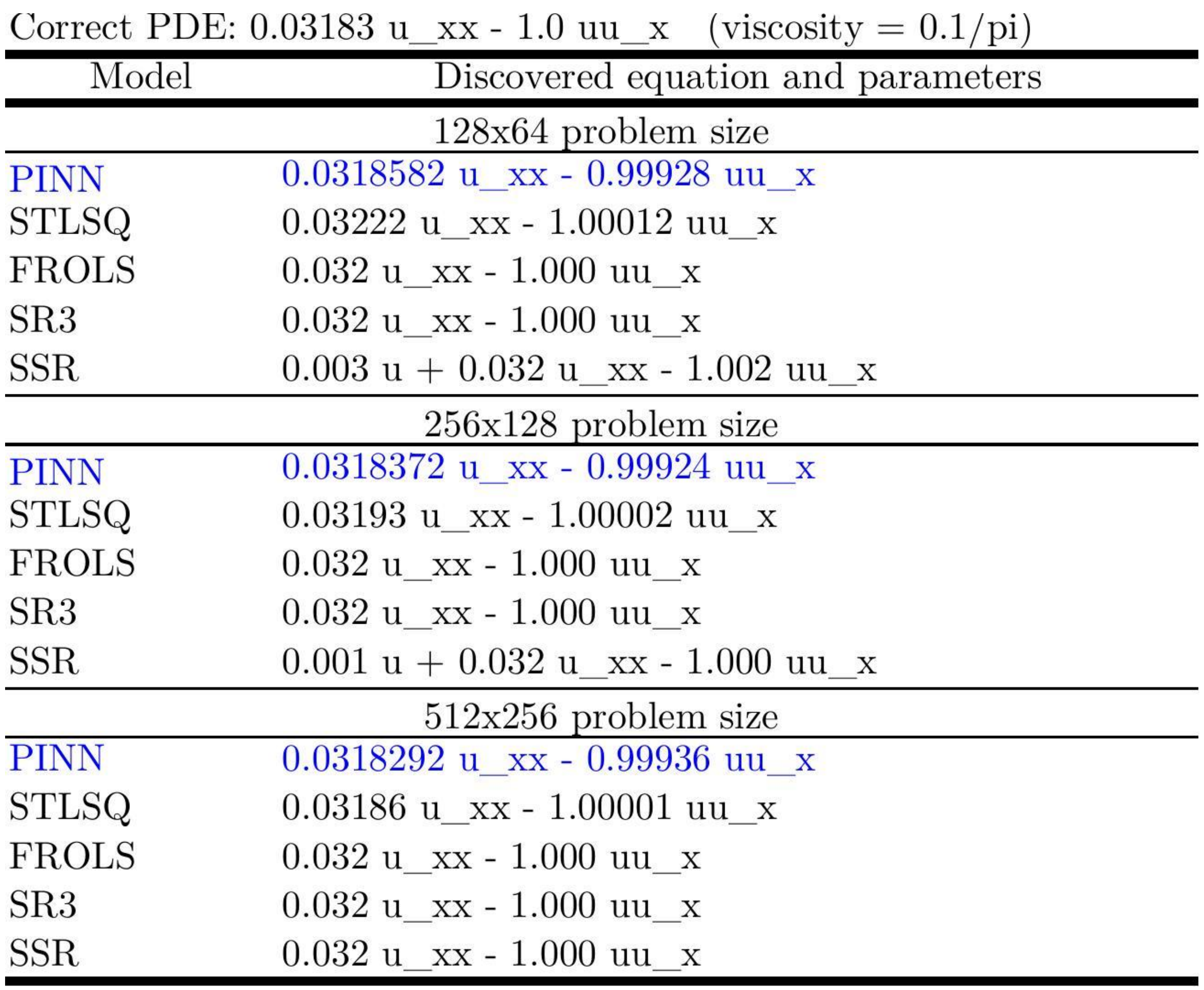
Parameter discovery results



Correct PDE: 0.003183 u_xx - 1.0 uu_x (viscosity = 0.01/pi)

Model	Discovered equation and parameters
	128x64 problem size
PINN	0.0033735 u_xx - 0.99912 uu_x
STLSQ	0.06420 u + 0.00505 u_xx - 1.06304 uu_x + + 0.00469 uuu_xx - 0.00001 uu_xxx
FROLS	-0.418 u
SR3	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
SSR	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
	256x128 problem size
PINN	0.0031779 u_xx - 0.99942 uu_x
STLSQ	0.00395 u_xx - 1.00869 uu_x + 0.00126 uuu_xx
FROLS	0.015 u + 0.004 u_xx - 1.003 uu_x
SR3	0.004 u_xx - 1.009 uu_x + 0.001 uuu_xx
SSR	0.011 u + 0.004 u_xx - 1.011 uu_x + 0.001 uuu_xx
	512x256 problem size
PINN	0.0031403 u_xx - 0.99850 uu_x
STLSQ	0.00339 u_xx - 1.00534 uu_x + 0.00041 uuu_xx
FROLS	0.006 u + 0.004 u_xx - 1.006 uu_x
SR3	0.003 u_xx - 1.005 uu_x
SSR	0.006 u + 0.003 u_xx - 1.007 uu_x

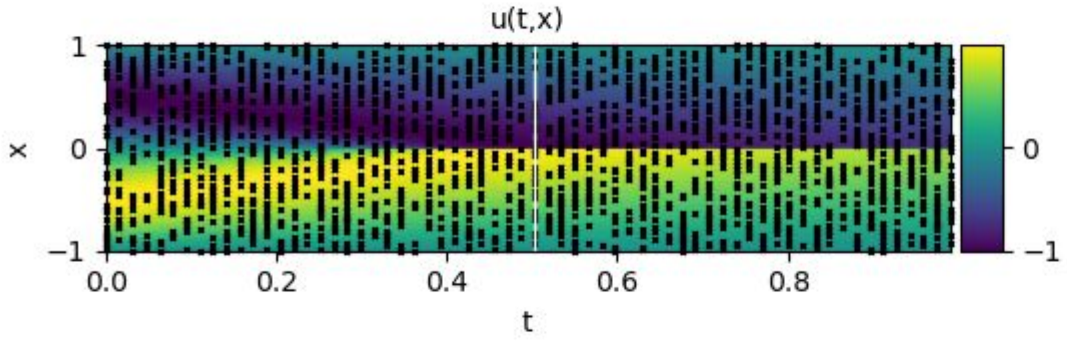
Parameter discovery results

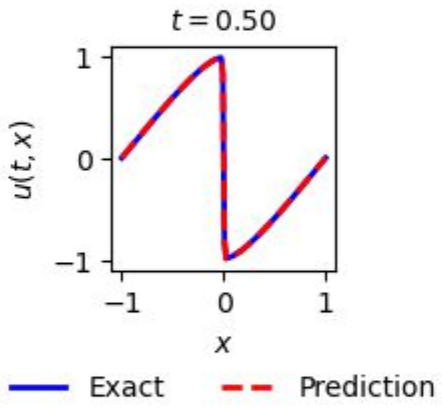


Correct PDE: 0.03183 u_xx - 1.0 uu_x (viscosity = 0.1/pi)

Model	Discovered equation and parameters
	128x64 problem size
PINN	0.0318582 u_xx - 0.99928 uu_x
STLSQ	0.03222 u_xx - 1.00012 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.003 u + 0.032 u_xx - 1.002 uu_x
	256x128 problem size
PINN	0.0318372 u_xx - 0.99924 uu_x
STLSQ	0.03193 u_xx - 1.00002 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.001 u + 0.032 u_xx - 1.000 uu_x
	512x256 problem size
PINN	0.0318292 u_xx - 0.99936 uu_x
STLSQ	0.03186 u_xx - 1.00001 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.032 u_xx - 1.000 uu_x

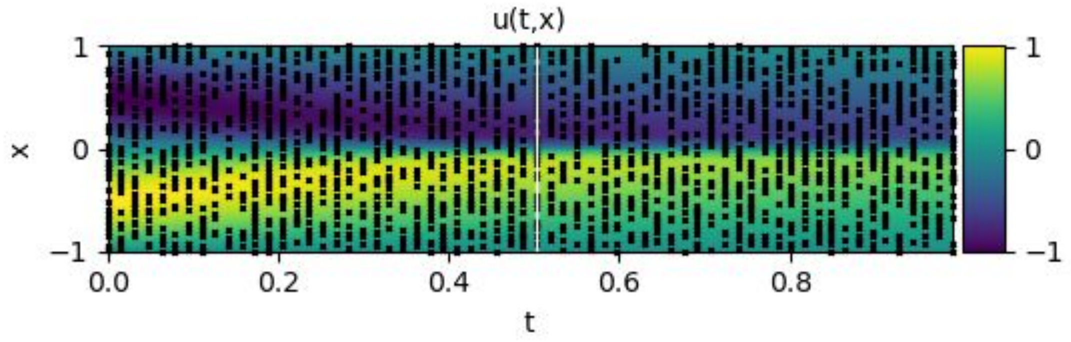
PINN visual assessment - 128x64, 0.01/π (low viscosity)

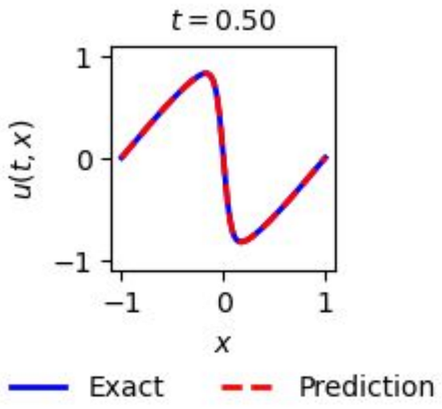




- PINN can accurately capture the non-linear behavior
  - SINDy would require a fine discretization for small viscosity values)
- The trained model is used for both discovery and solution
- No discretization of the spatio-temporal domain
- 2,000 CPs for training

PINN visual assessment - 128x64, 0.1/π (high viscosity)

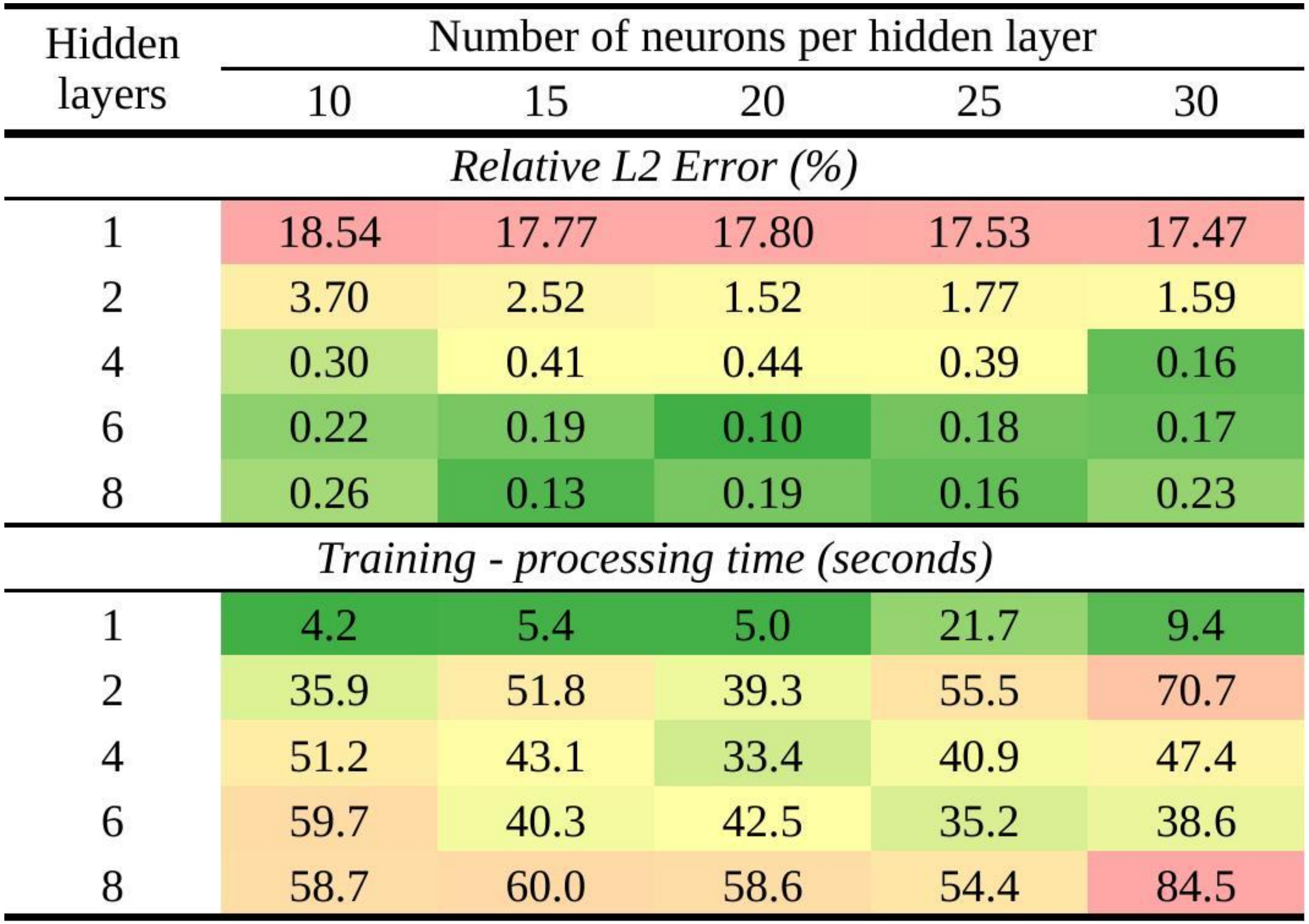




- PINN perform accurately
    - SINDy can accurately capture as well

L2 error and training time [s] X Hidden Layers

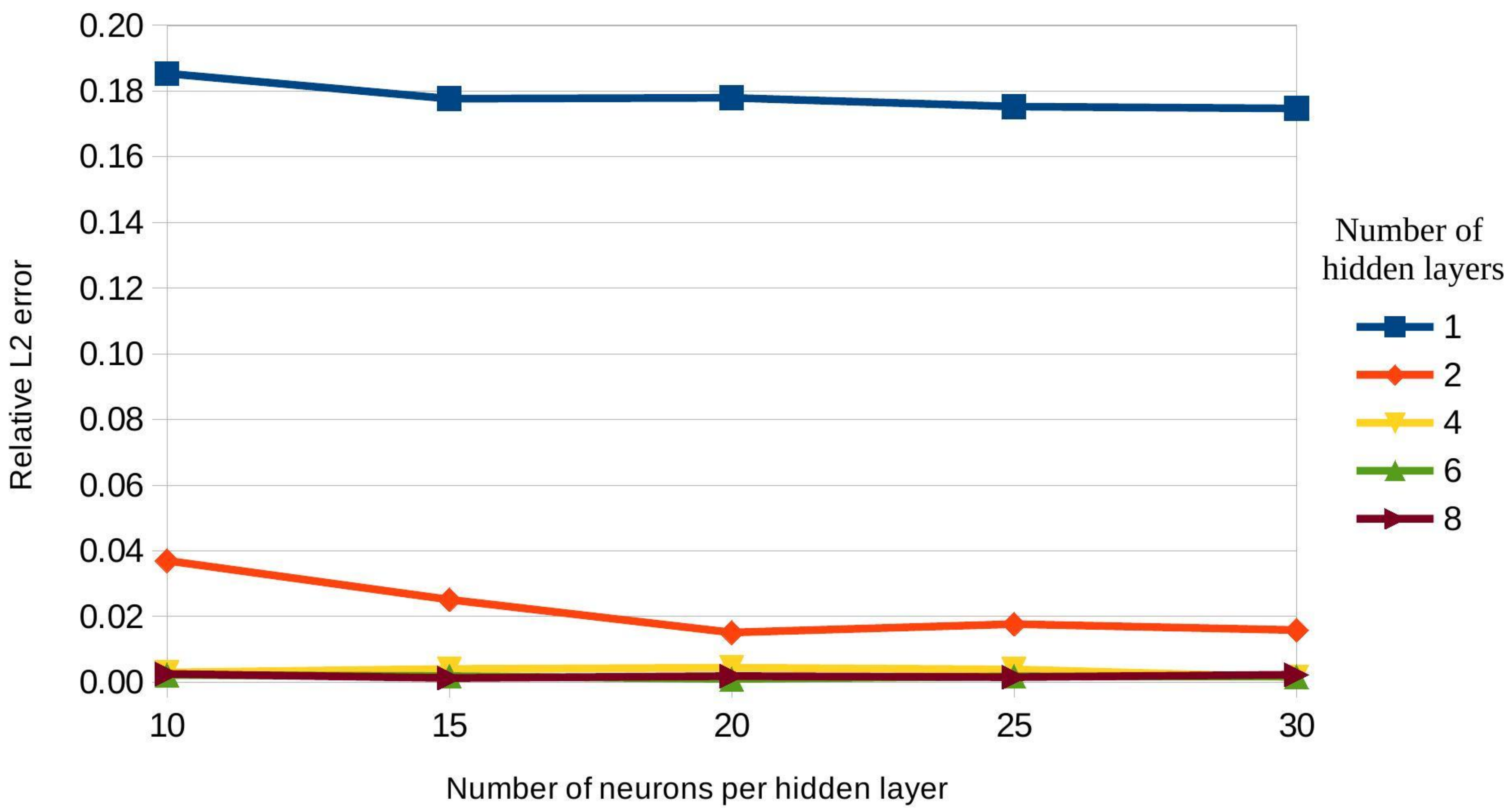
(SDumont Sequana X1129)



Hidden layers	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
1	18.54	17.77	17.80	17.53	17.47
2	3.70	2.52	1.52	1.77	1.59
4	0.30	0.41	0.44	0.39	0.16
6	0.22	0.19	0.10	0.18	0.17
8	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
1	4.2	5.4	5.0	21.7	9.4
2	35.9	51.8	39.3	55.5	70.7
4	51.2	43.1	33.4	40.9	47.4
6	59.7	40.3	42.5	35.2	38.6
8	58.7	60.0	58.6	54.4	84.5

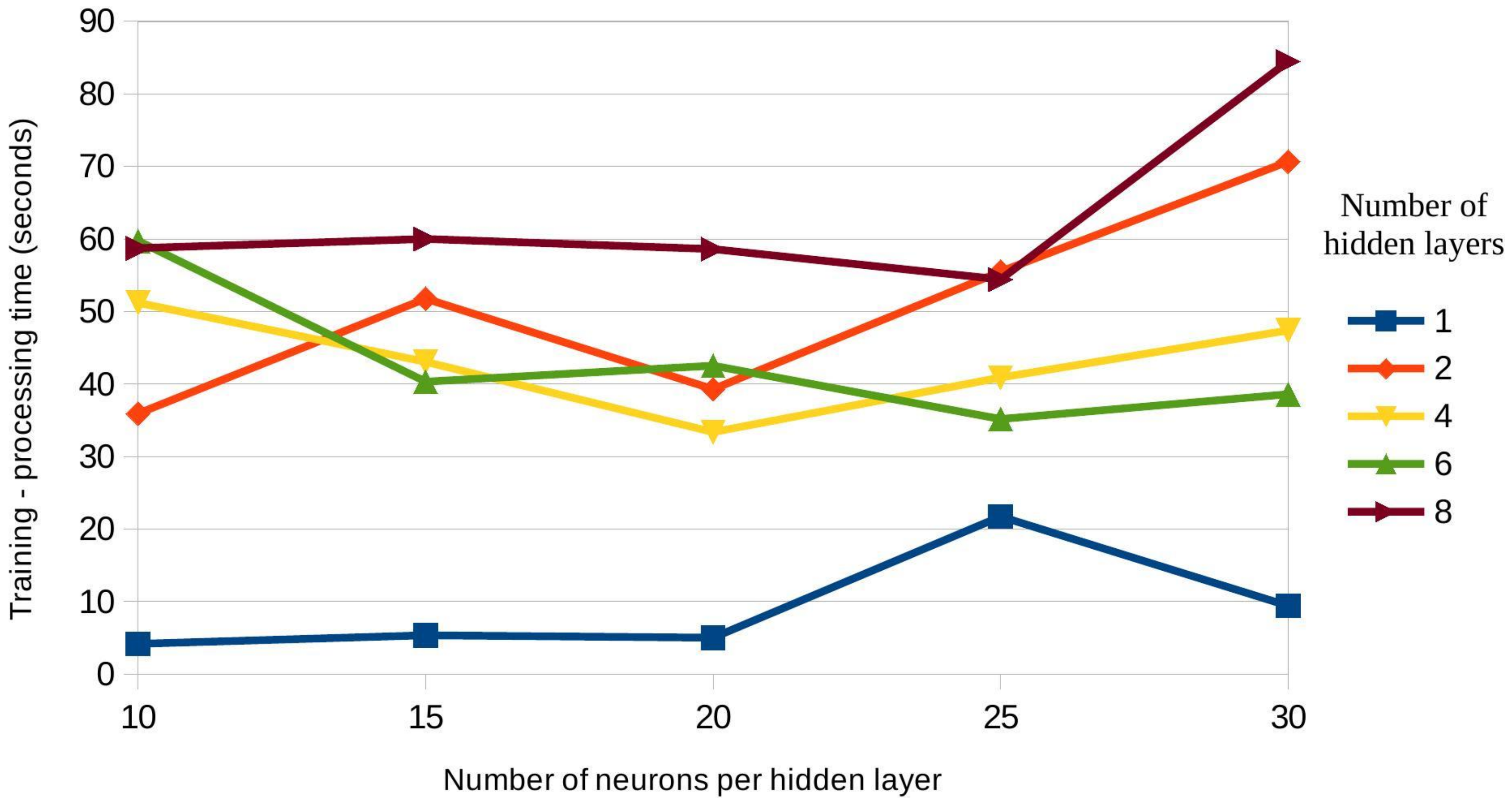
Relative L2 error (%)

(SDumont Sequana X1129)



Processing time [s]

(SDumont Sequana X1129)



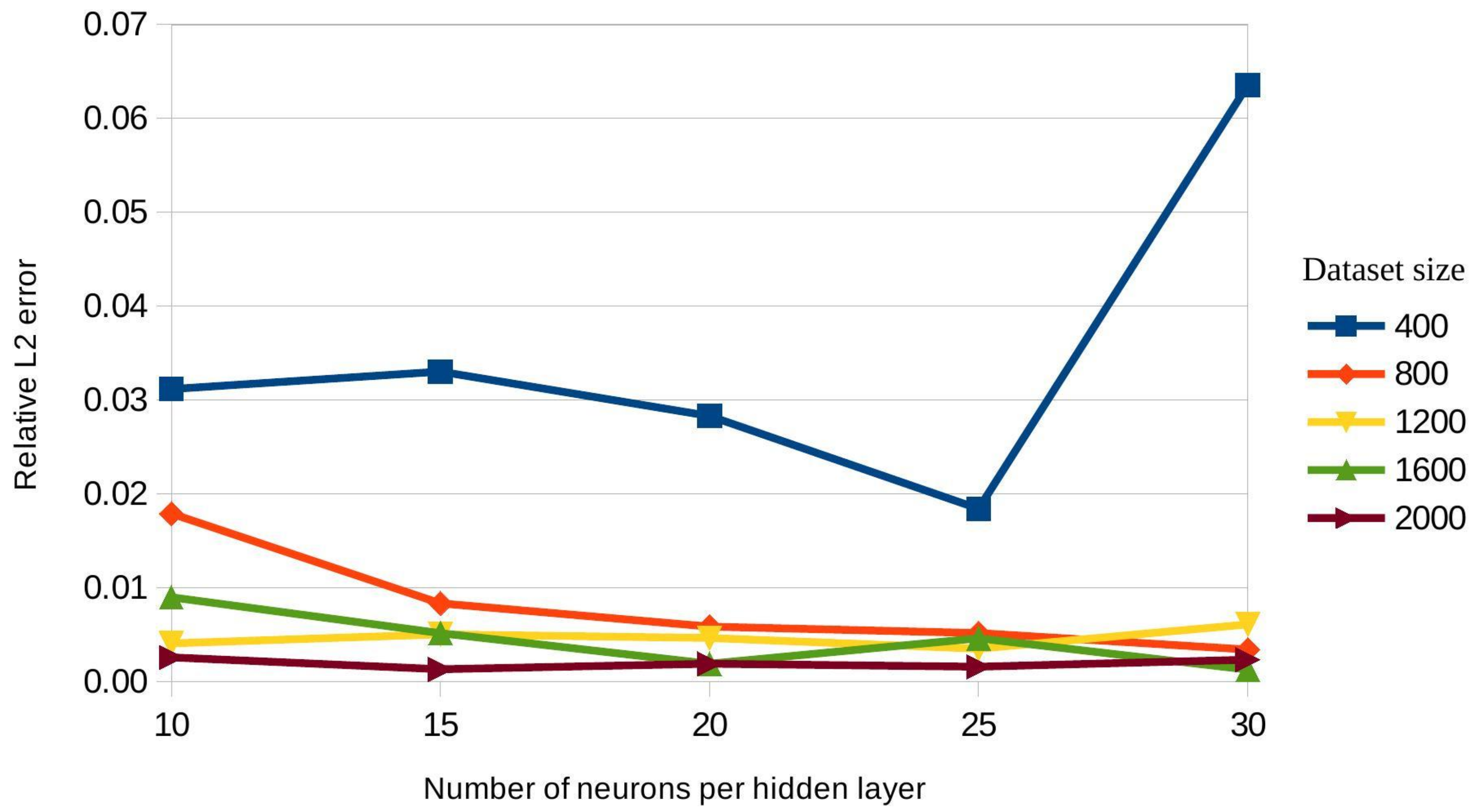
L2 error and training time [s] X CP size

(SDumont Sequana X1129)

Dataset size	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
400	3.12	3.30	2.83	1.84	6.36
800	1.79	0.83	0.59	0.52	0.34
1200	0.41	0.50	0.46	0.35	0.61
1600	0.90	0.51	0.19	0.46	0.13
2000	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
400	57.9	82.3	83.3	59.8	58.6
800	79.7	53.5	63.2	45.0	63.0
1200	63.7	52.2	43.8	42.1	56.8
1600	59.9	27.5	45.3	46.5	56.4
2000	58.7	60.0	58.6	54.4	84.5

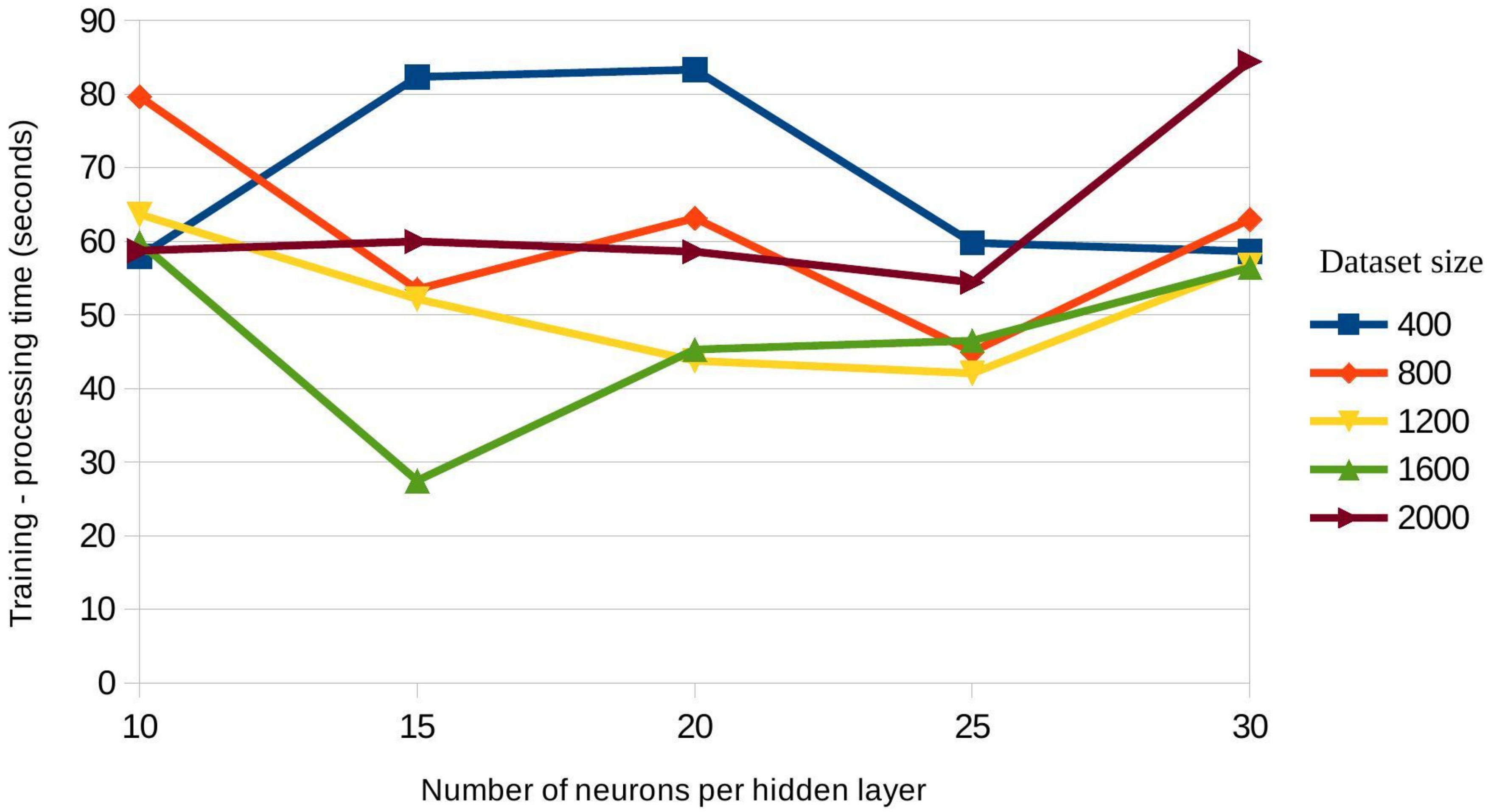
Relative L2 error (%)

(SDumont Sequana X1129)



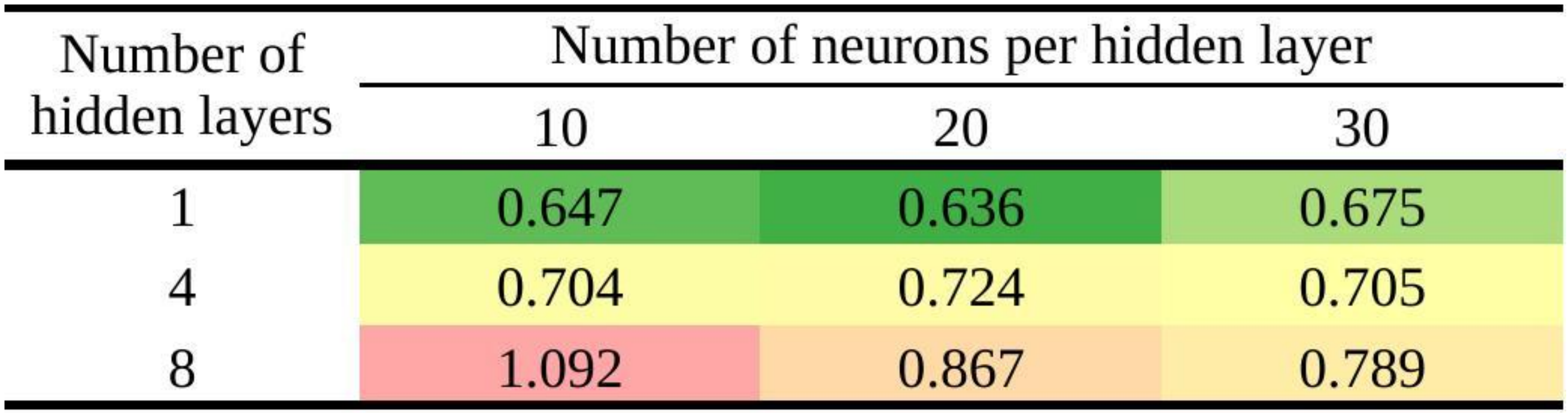
Processing time [s]

(SDumont Sequana X1129)



Prediction time [s]

(SDumont Sequana X1129)



Number of hidden layers	Number of neurons per hidden layer		
	10	20	30
1	0.647	0.636	0.675
4	0.704	0.724	0.705
8	1.092	0.867	0.789

(Complements the previous graphs)

5

Final Considerations

Final considerations

- PINN is an innovative and promising approach that combines data-driven and physics-based strategies
- Toy problem (1D Burgers' Equation) with known exact solution
- Comparison of PINN with 4 SINDy versions
- PINN assessment of accuracy and performance

Future works

- Explore new PINN-based approaches for 2D/3D inverse and direct problems
- Choose a new test case related to an INPE application (ecRad radiation module in weather forecast model?)
- Real-world problems with limited-size or noisy datasets
- Use of GPU frameworks like Modulus and Horovod

Thanks!

Source code: http://github.com/efurlanm/pd1b24/

Eduardo Furlan Miranda. Applied Computing Post-graduation Program (CAP/INPE). <efurlanm@gmail.com>.

Stephan Stephany. Coordination of Applied Research and Technological Development (COPDT/INPE). <stephan.stephany@inpe.br>.

Roberto Pinto Souto. National Laboratory for Scientific Computing (LNCC). <rpsouto@lncc.br>.