EXCALIBUR 10

SiMLInt

Simulation and Machine Learning Integration

05. 06. 2023

Moritz Linkmann[1], Iakovos Panourgias[2]

[1]School of Mathematics and Maxwell Institute for Mathematical Sciences

[2]Edinburgh Parallel Computing Centre

University of Edinburgh

Met Office

UKRI UK Research and Innovation

UK Atomic Energy Authority

SiMLInt

ExCALIBUR cross-cutting project

People

Amy Krause (EPCC)
Jacob Page (School of Mathematics)
Anna Roubíčková (EPCC)
Iakovos Pangourias (EPCC)
Elena Breitmoser (EPCC)

Cross-cutting theme

A coordinated approach addressing a known technology or infrastructure issue, which, if resolved, will lead to significant progress across a range of exascale software development challenges

Developed by EPCC and the School of Mathematics at the University of Edinburgh, UK, SiMLInt provides infrastructure that allows embedding Machine Learning capability to large-scale numerical simulations.

More information

https://excalibur.ac.uk/
https://excalibur.ac.uk/projects/simlint/

Contact us

simlint@mlist.is.ed.ac.uk

Motivation

$$\partial_t u_i + \partial_j \left(u_i u_j + p\delta_{ij} - \frac{1}{Re} S_{ij} \right) = f_i$$

fully resolved simulation → ground truth

Aim → simulation on coarser grid

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell - \tau_{ij}^\ell \right) = \overline{f}_i^\ell$$

$$\tau_{ij}^\ell = \overline{u_i u_j}^\ell - \overline{u}_i^\ell \overline{u}_j^\ell$$

subgrid-scale stress tensor

$$\tau_{ij}^{\text{model}} = (\ell C_S)^2 \sqrt{\overline{S}_{ij}^\ell \overline{S}_{ij}^\ell} \overline{S}_{ij}^\ell$$

Smagorinsky parametrisation

$$\tau_{ij}^{\text{model}} = f(a, b, c, \ldots) \overline{S}_{ij}^\ell$$

multiple parameters → ML

$$\tau_{ij}^{\text{model}} = X_{ij}(a, b, c, \ldots)$$

learn subgrid stresses

Motivation

→ **solve equation with model term on a coarser grid**

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell - \tau_{ij}^{\text{model}} \right) = \overline{f}_i^\ell$$

$$\tau_{ij}^{\text{model}} = (\ell C_S)^2 \sqrt{\overline{S}_{ij}^\ell \overline{S}_{ij}^\ell} \overline{S}_{ij}^\ell$$ Smagorinsky parametrisation

$$\tau_{ij}^{\text{model}} = f(a, b, c, \ldots) \overline{S}_{ij}^\ell$$ multiple parameters → ML

$$\tau_{ij}^{\text{model}} = X_{ij}(a, b, c, \ldots)$$ learn subgrid stresses

Motivation

→ **solve equation with *correction* term on a coarser grid**

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell \right) = \overline{f}_i^\ell + F_j^{\text{error}}$$

$$F_j^{\text{error}} = X_j(\overline{\boldsymbol{u}}^\ell, \boldsymbol{u})$$

learned corrections

Motivation

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell - \tau_{ij}^{\text{model}} \right) = \overline{f}_i^\ell$$

or

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell \right) = \overline{f}_i^\ell + F_j^{\text{error}}$$

$\tau_{ij}^{\text{model}} = (\ell C_S)^2 \sqrt{\overline{S}_{ij}^\ell \overline{S}_{ij}^\ell} \overline{S}_{ij}^\ell$	Smagorinsky parametrisation
$\tau_{ij}^{\text{model}} = f(a, b, c, \ldots) \overline{S}_{ij}^\ell$	multiple parameters → ML
$\tau_{ij}^{\text{model}} = X_{ij}(a, b, c, \ldots)$	learn subgrid stresses
$F_j^{\text{error}} = X_j(\overline{\boldsymbol{u}}^\ell, \boldsymbol{u})$	learned corrections

Motivation

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell - \tau_{ij}^{\text{model}} \right) = \overline{f}_i^\ell$$

or

$$\partial_t \overline{u}_i^\ell + \partial_j \left(\overline{u}_i^\ell \overline{u}_j^\ell + \overline{p}^\ell \delta_{ij} - \frac{1}{Re} \overline{S}_{ij}^\ell \right) = \overline{f}_i^\ell + F_j^{\text{error}}$$

Aim → provide infrastructure for any of these data-driven approaches

→ couple BOUT++ with ML-model(s)

$$\tau_{ij}^{\text{model}} = f(a, b, c, \ldots) \overline{S}_{ij}^\ell$$ multiple parameters → ML

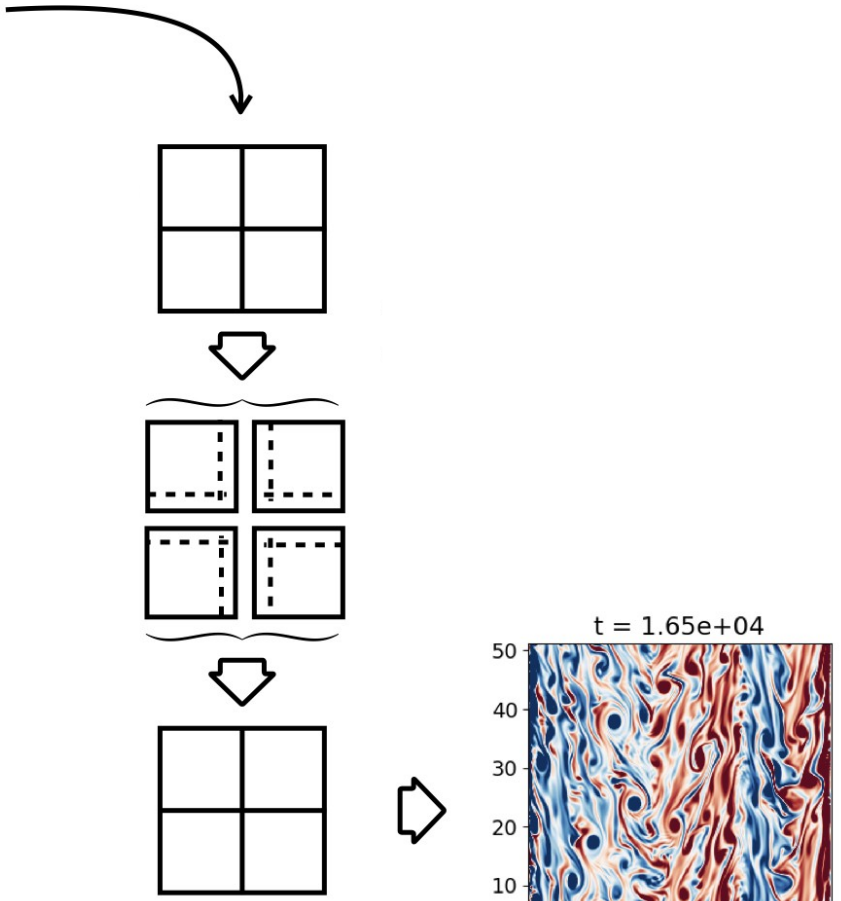
$$\tau_{ij}^{\text{model}} = X_{ij}(a, b, c, \ldots)$$ learn subgrid stresses

$$F_j^{\text{error}} = X_j(\overline{\boldsymbol{u}}^\ell, \boldsymbol{u})$$ learned corrections

EXCALIBUR

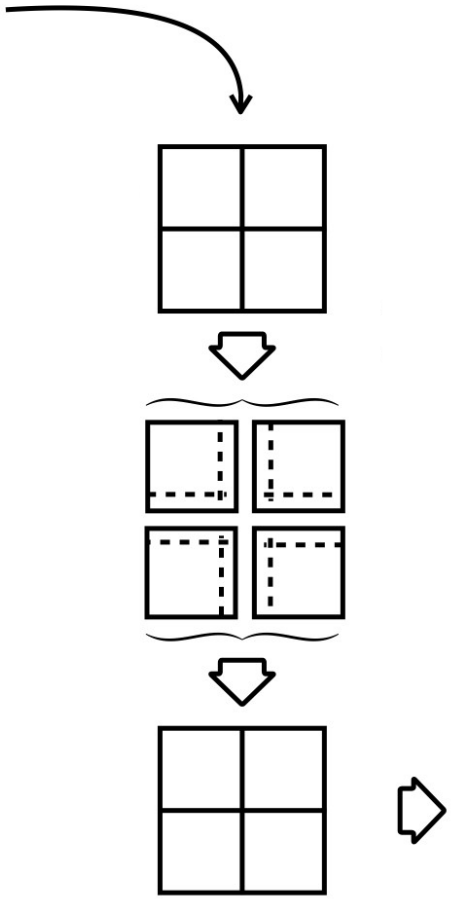
Concept and Aims

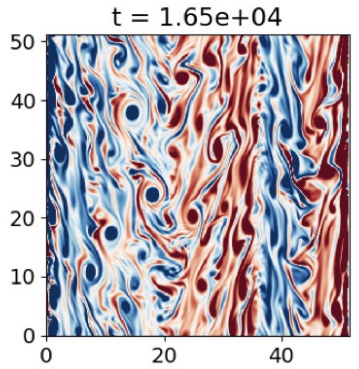
$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$



Concept and Aims

$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$





Objectives

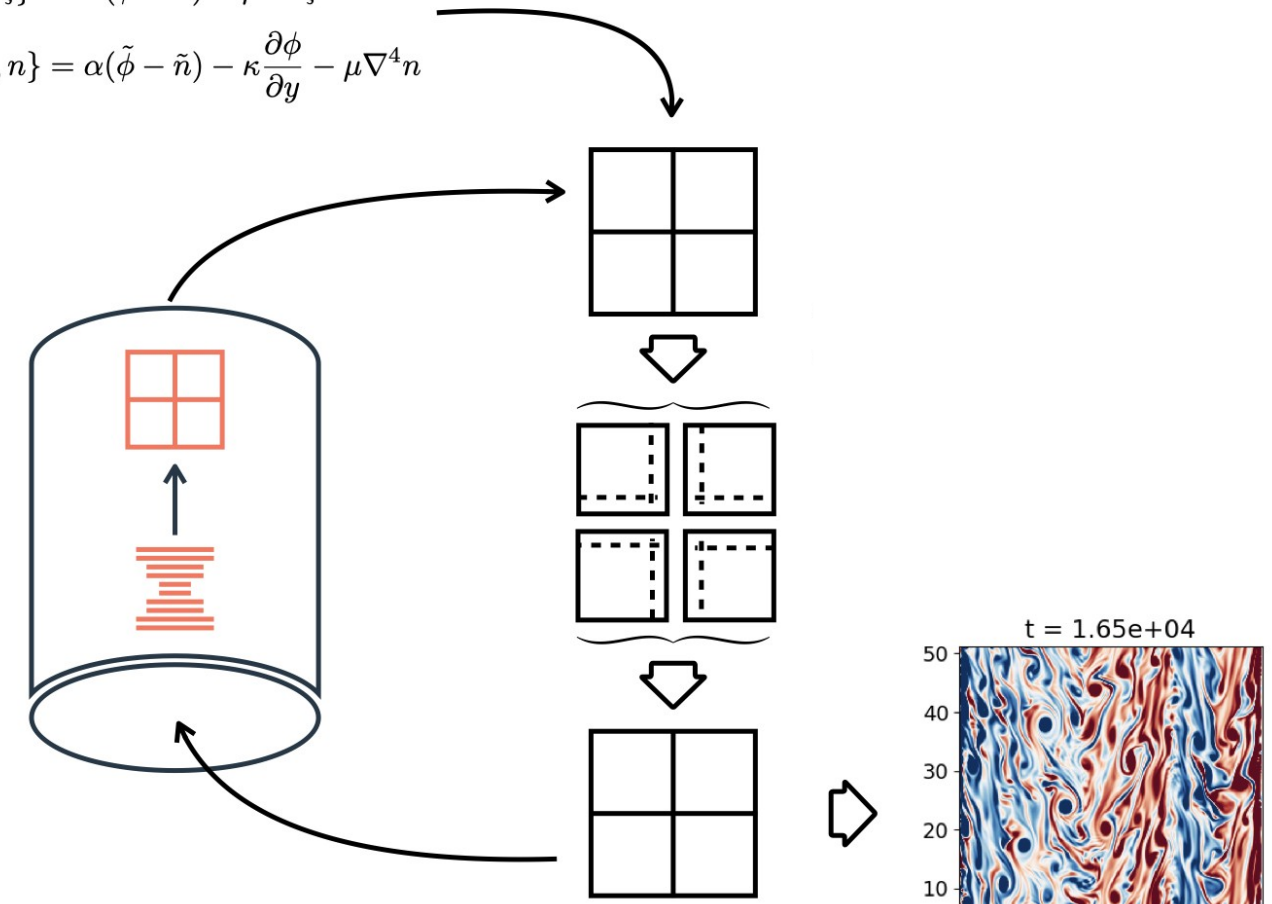
Provide infrastructure for coupling BOUT++ / ML

building problem-specific CNN architectures

generation of training data

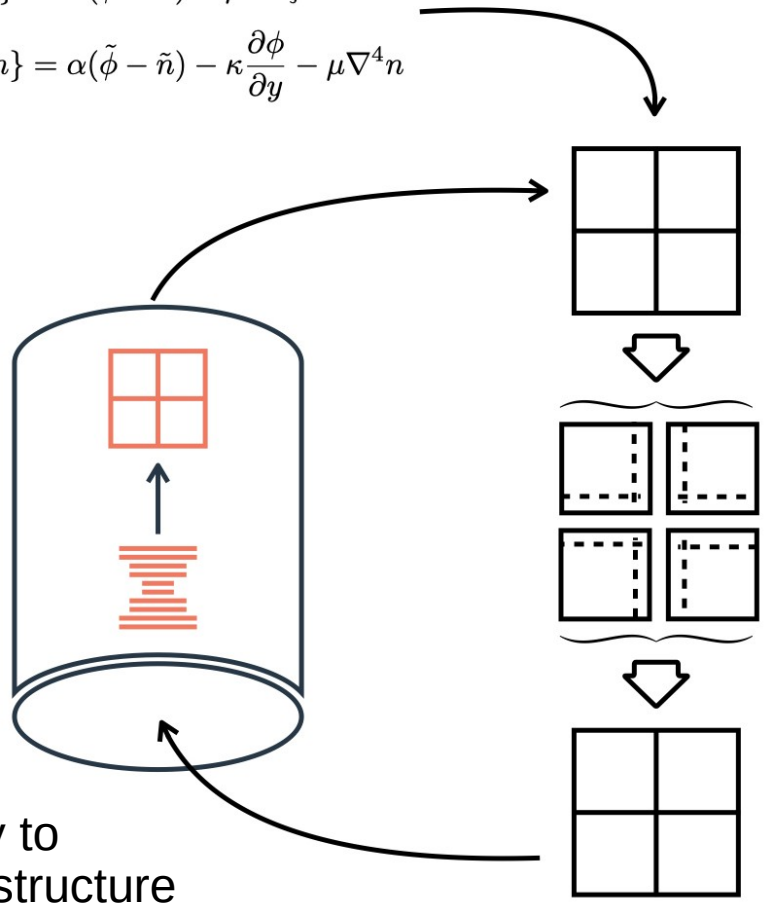
Concept and Aims

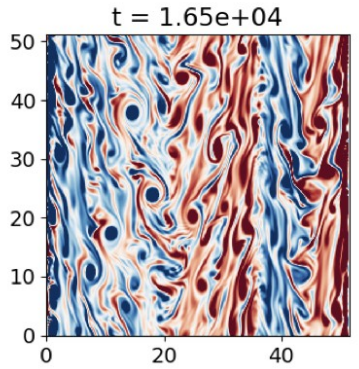
$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$



Concept and Aims

$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$



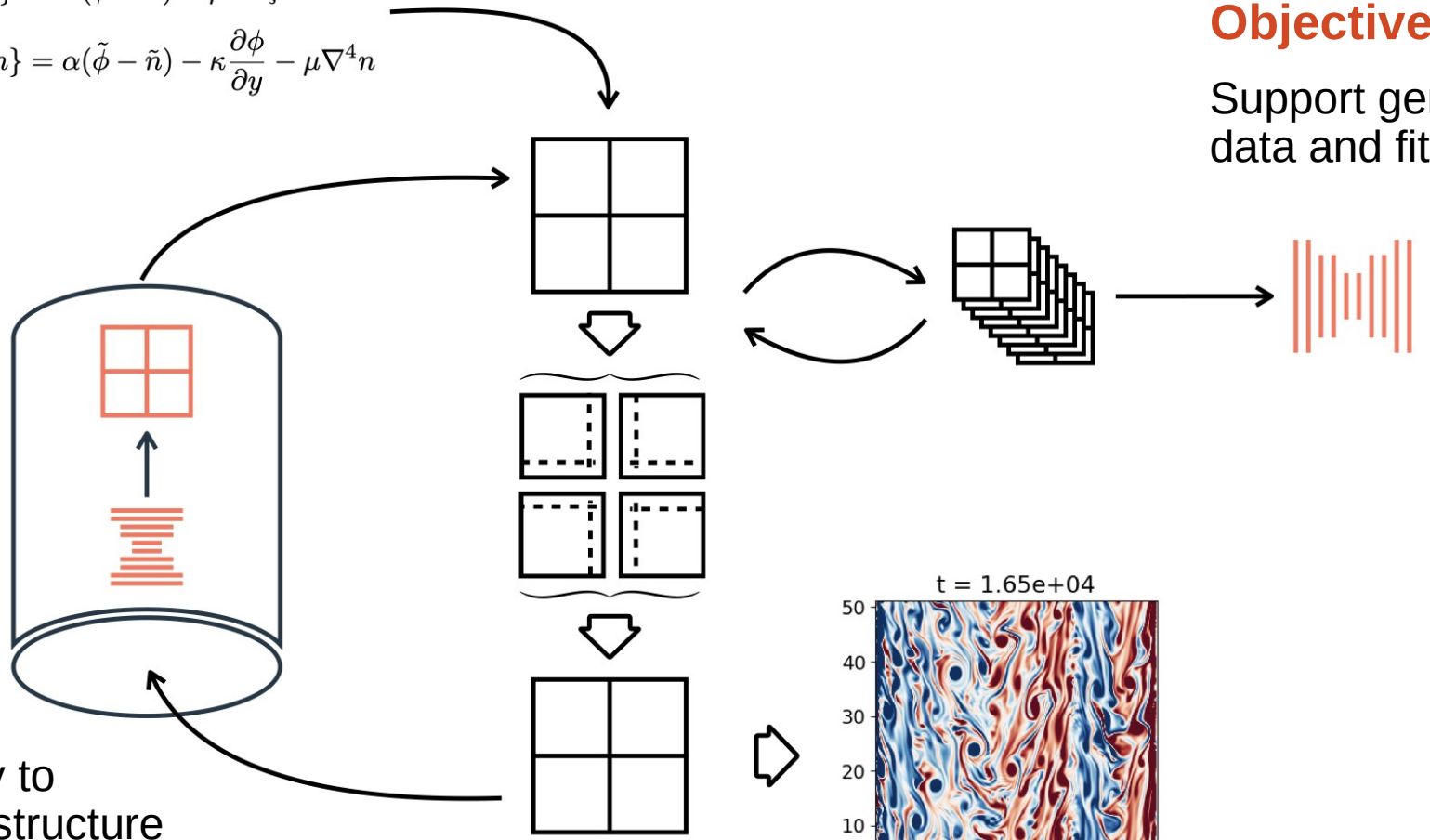


Objective 1:

Develop a library to provide the infrastructure

Concept and Aims

$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$



Objective 2:

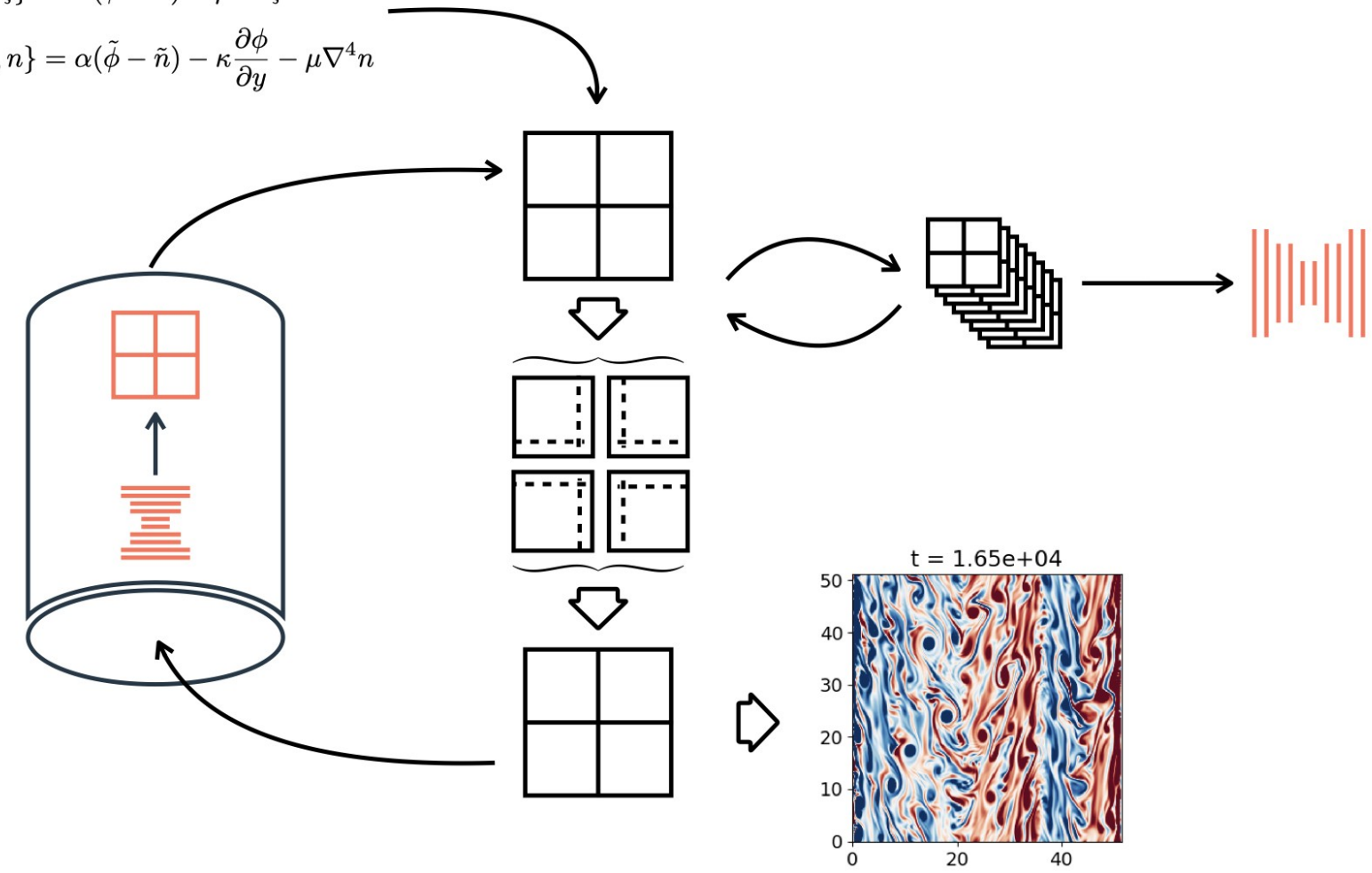
Support generation of training data and fitting the ML model

Objective 1:

Develop a library to provide the infrastructure

Identified tools/technologies

$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$

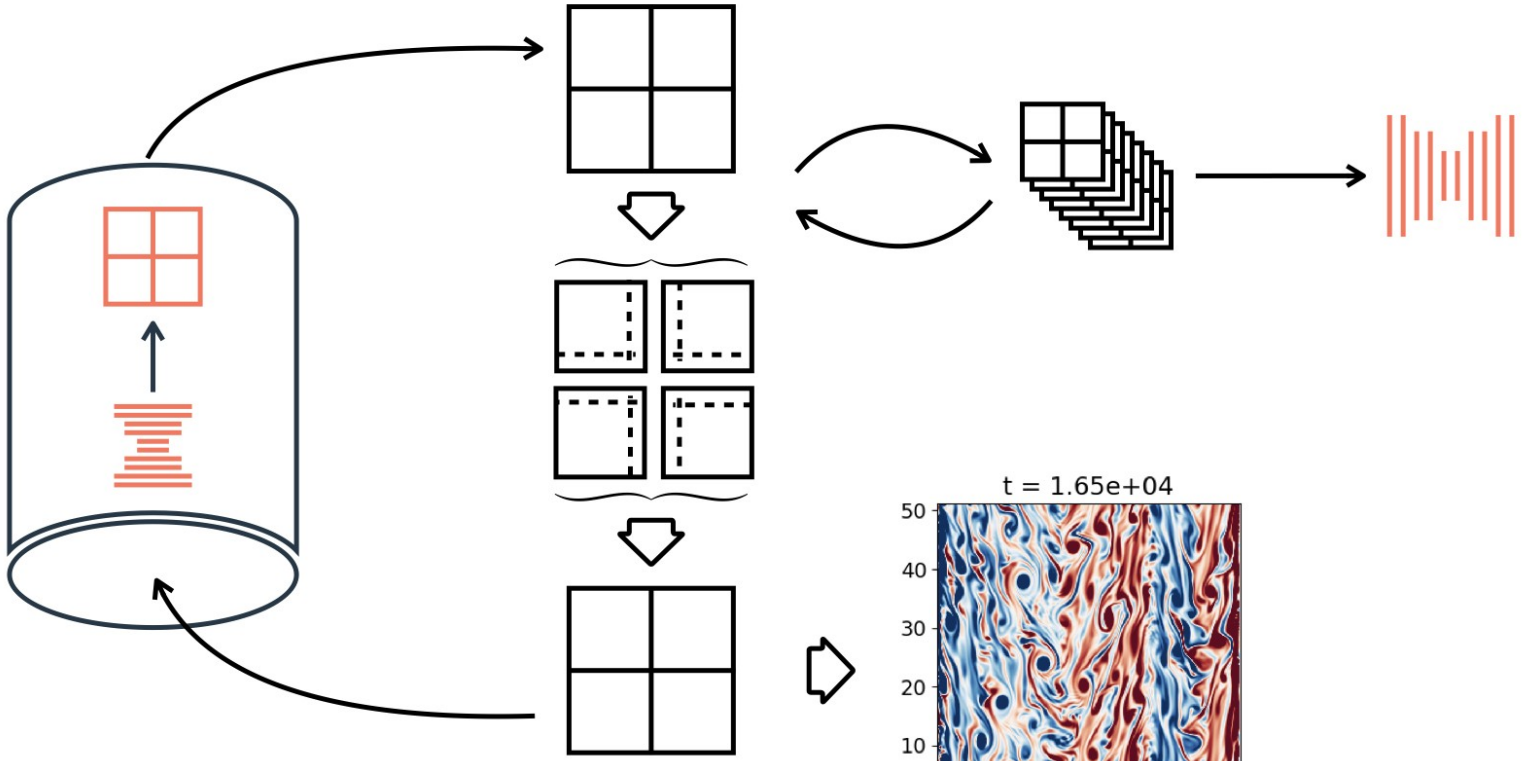


Identified tools/technologies

$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$

$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$

Hasegawa-Wakatani

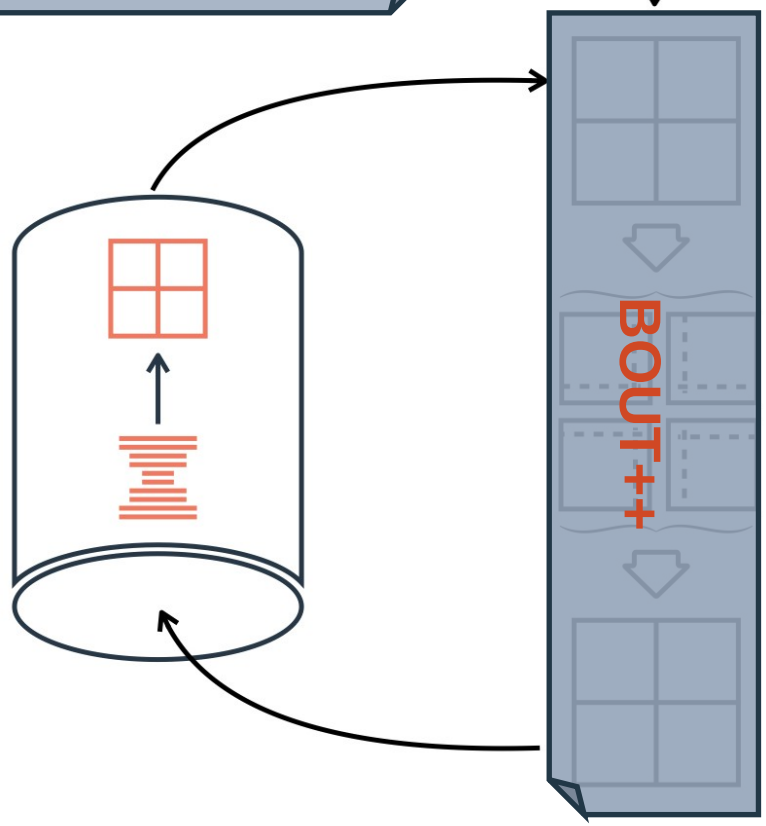


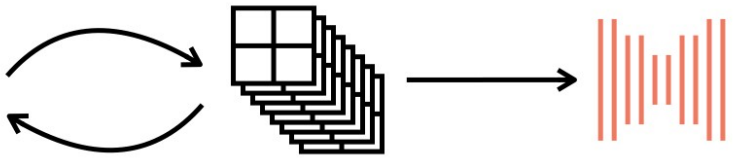
Identified tools/technologies

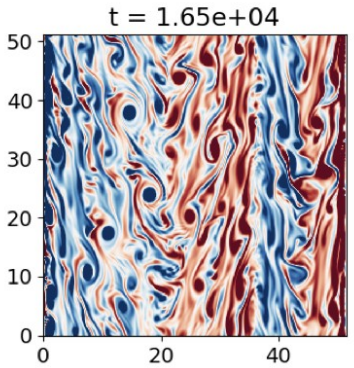
$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$

$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\phi - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$

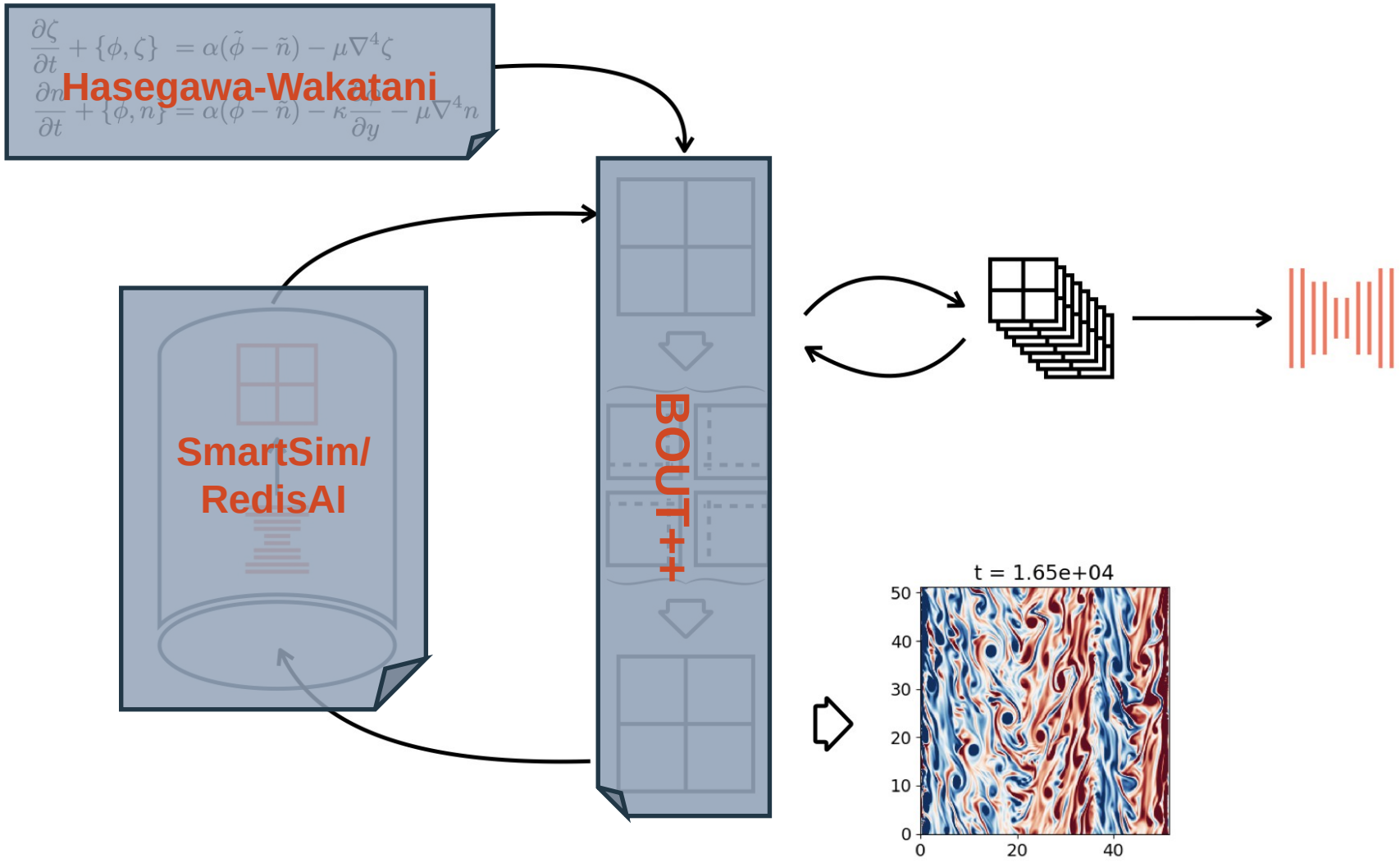
Hasegawa-Wakatani







Identified tools/technologies



Identified tools/technologies

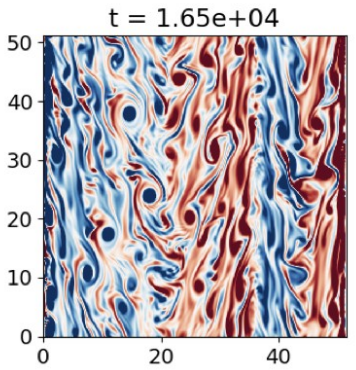
$$\frac{\partial \zeta}{\partial t} + \{\phi, \zeta\} = \alpha(\tilde{\phi} - \tilde{n}) - \mu \nabla^4 \zeta$$
$$\frac{\partial n}{\partial t} + \{\phi, n\} = \alpha(\tilde{\phi} - \tilde{n}) - \kappa \frac{\partial \phi}{\partial y} - \mu \nabla^4 n$$

Hasegawa-Wakatani

SmartSim/
RedisAI

BOUT++

Python/TensorFlow
bash



SmartSim

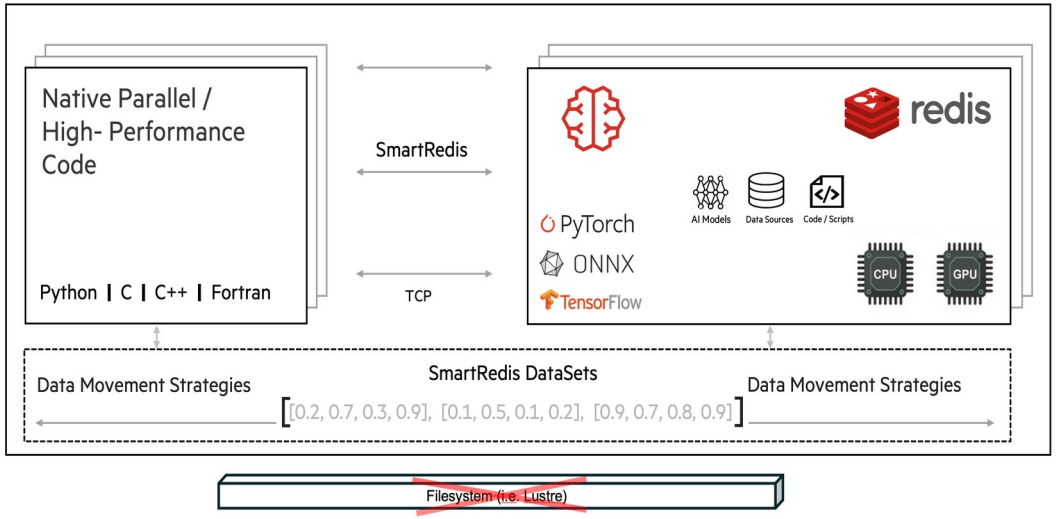
github.com/CrayLabs/SmartSim

- A library to facilitate data exchange between HPC simulations and Machine Learning workflows

- Two main components:

  1. **SmartSim**: The infrastructure to call Python ML models from Fortran, C and C++ (and Python)

  2. **SmartRedis**: A collection of clients to exchange data (tensors, models) with a distributed, in-memory database (Redis cluster) across multiple compute nodes



From: https://www.craylabs.org/docs/overview.html

- Provides uniform API for pulling/pushing simulation data from C, C++ and Fortran at run-time

- Orchestration ability to coordinate the runs of simulation and machine learning in distributed environment

SmartSim (Infrastructure Library)

automate process of deploying HPC workloads alongside **in-memory database**

deploy **distributed**, shared-nothing, **in-memory cluster of Redis instances** across compute nodes
→ **Orchestrator**

Orchestrator + HPC applications
→ connect workloads (e.g. trained ML models) to other applications with SmartRedis clients

- API to start, monitor, and stop HPC jobs from Python script / Jupyter notebook.
- Automated deployment of in-memory data staging (Redis) and computational storage (RedisAI).
- Programmatic launches of batch and in-allocation jobs on PBS, Slurm, LSF, and Cobalt systems
- Creating and configuring ensembles of workloads with isolated communication channels.

SmartRedis (Client Library)

collection of Redis clients supporting RedisAI capabilities and include additional features for HPC applications

Key features of **RedisAI** supported by SmartRedis

- A tensor data type in Redis
- TensorFlow, TensorFlow Lite, Torch, and ONNXRuntime backends for model evaluations
- TorchScript storage and evaluation

Key features of **SmartRedis** developed for large, distributed HPC architectures

- Redis cluster support for RedisAI data types (tensors, models, and scripts).
- Distributed model and script placement for parallel evaluation that **maximizes hardware utilisation** and throughput
- A **DataSet storage format** to **aggregate multiple tensors** and metadata into a **single Redis cluster** hash slot to **prevent data scatter** on Redis clusters and maintain contextual relationships between tensors. Useful when clients produce tensors and metadata that are referenced or utilised together.
- An **API for efficiently aggregating DataSet objects** distributed on one or more database nodes.
- **Compatibility with SmartSim ensemble capabilities** to prevent key collisions with tensors, DataSet, models, and scripts when clients are part of an ensemble of applications.

Collaborations and links

1. BOUT++ users with simulations

2. Similar problems/simulations, other numerical solvers

3. Machine Learning specialists and Data Scientists
   – errors, visualisations, uncertainty, verification/validation of models, …

simlint@mlist.is.ed.ac.uk

Thanks for listening!

Questions?

References

- D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner and S. Hoyer: **Machine learning–accelerated computational fluid dynamics.** Proceedings of the National Academy of Sciences, 118(21), 2021

- Maulik, R., San, O., Rasheed, A., & Vedula, P. (2019). **Subgrid modelling for two-dimensional turbulence using neural networks.** Journal of Fluid Mechanics, 858, 122-144. doi:10.1017/jfm.2018.770

- P. Watson: **Applying machine learning to improve simulations of a chaotic dynamical system using empirical error correction.** Journal of Advances in Modeling Earth Systems, May 2019.

- M. Wakatani and A. Hasegawa: **A collisional drift wave description of plasma edge turbulence.** The Physics of fluids, 27(3):611–618, 1984

- A. Hasegawa and M. Wakatani. **Self-organization of electrostatic turbulence in a cylindrical plasma.** Phys. Rev. Lett., 59:1581–1584, Oct 1987.

- TensorFlow: `tensorflow.org`
  Keras: `keras.io`

- S. Partee, M. Ellis, A. Rigazzi, S. Bachman, G. Marques, A. Shao, and B. Robbins: **Using machine learning at scale in HPC simulations with smartsim: An application to ocean climate modeling.** CoRR, abs/2104.09355, 2021. `github.com/CrayLabs/SmartSim`