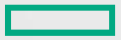
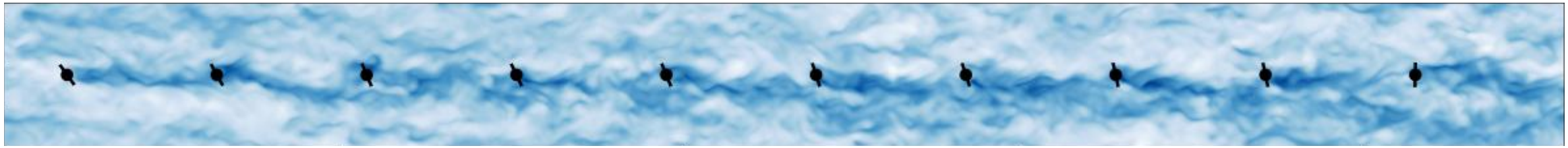
Use cases and examples



Dynamic turbine wake steering in wind farms

Dynamic turbine wake steering in wind farms



Reference: Project is work in progress, Andrew Mole, Imperial College London

Turbine wakes can reduce power output of downstream turbines

We can steer the wake by adjusting angle (yaw) of turbines

In general, the system is very complex, changing direction/speed, multiple turbines, interaction of wakes etc.

Use reinforcement learning to find some optimal set of angles to optimize power output

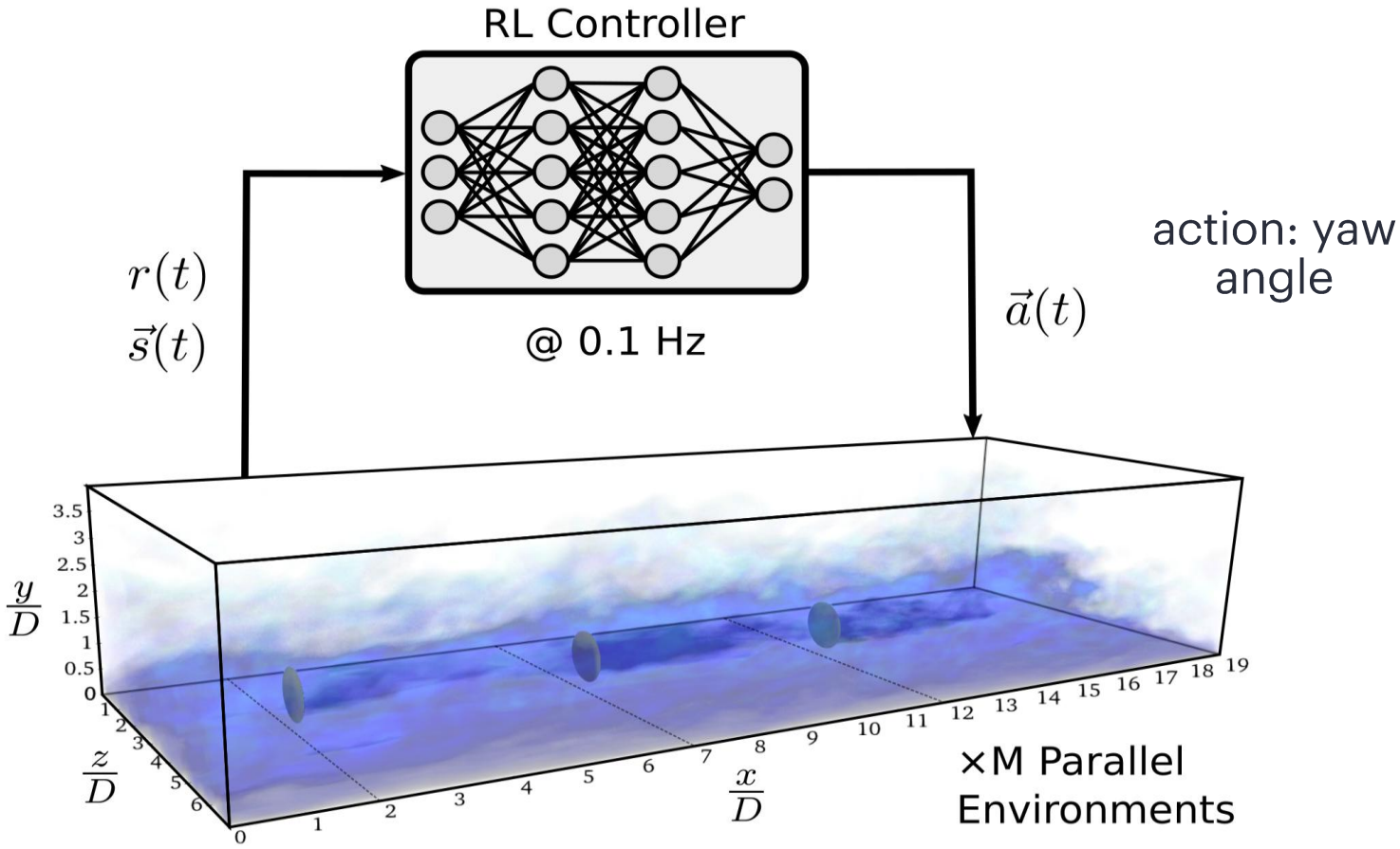
Direct coupling with SmartSim an improvement over file-based coupling

https://arxiv.org/abs/2506.20554

Reinforcement learning

reward: power output

state: flow measurement



Environment: large Eddy simulations (LES)

x M different ensembles

Implementation

Initial approach was to couple the Large Eddy simulation of the windfarm with pytorch reinforcement learning model

Used file storage to communicate (a_t s_t r_t)

Issues:

- Large files often written was costly
- Was not going to be practical when simulating larger wind farms
- Made it difficult to train multiple environments at a time

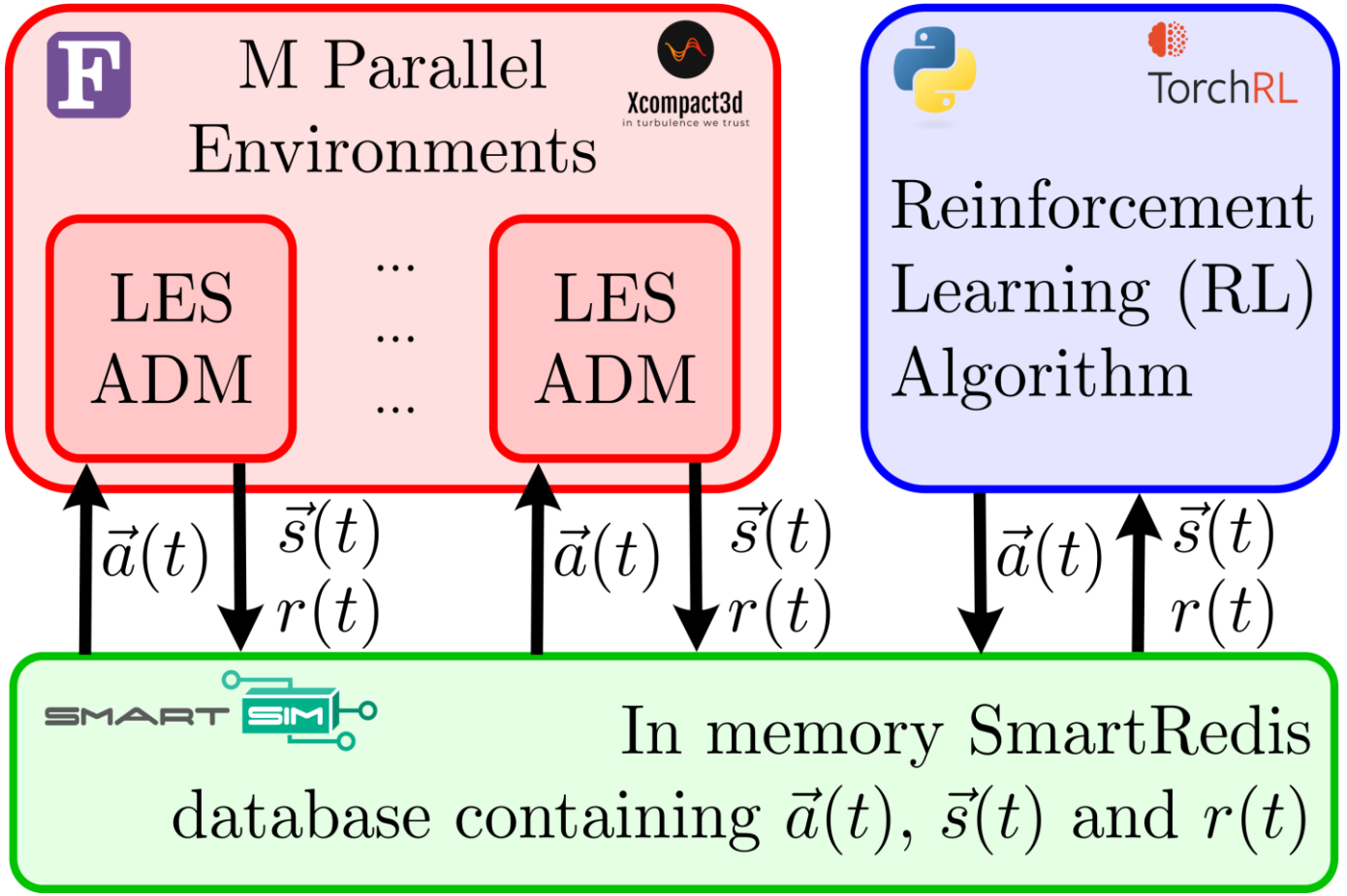
Moved to SmartSim

— Now using SmartRedis database

Modifications needed

- XCompact3D Replace file ops with calls to send and retrieve data to SmartRedis
- Configured SmartSim to start XCompact3D environments/simulations (32 at a time for example)

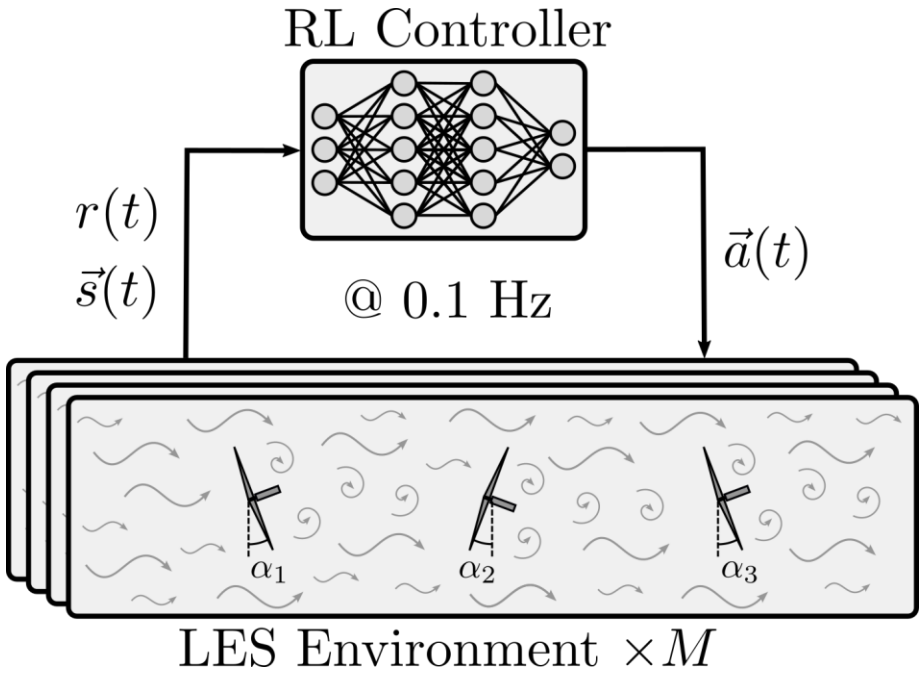
SmartSim usage



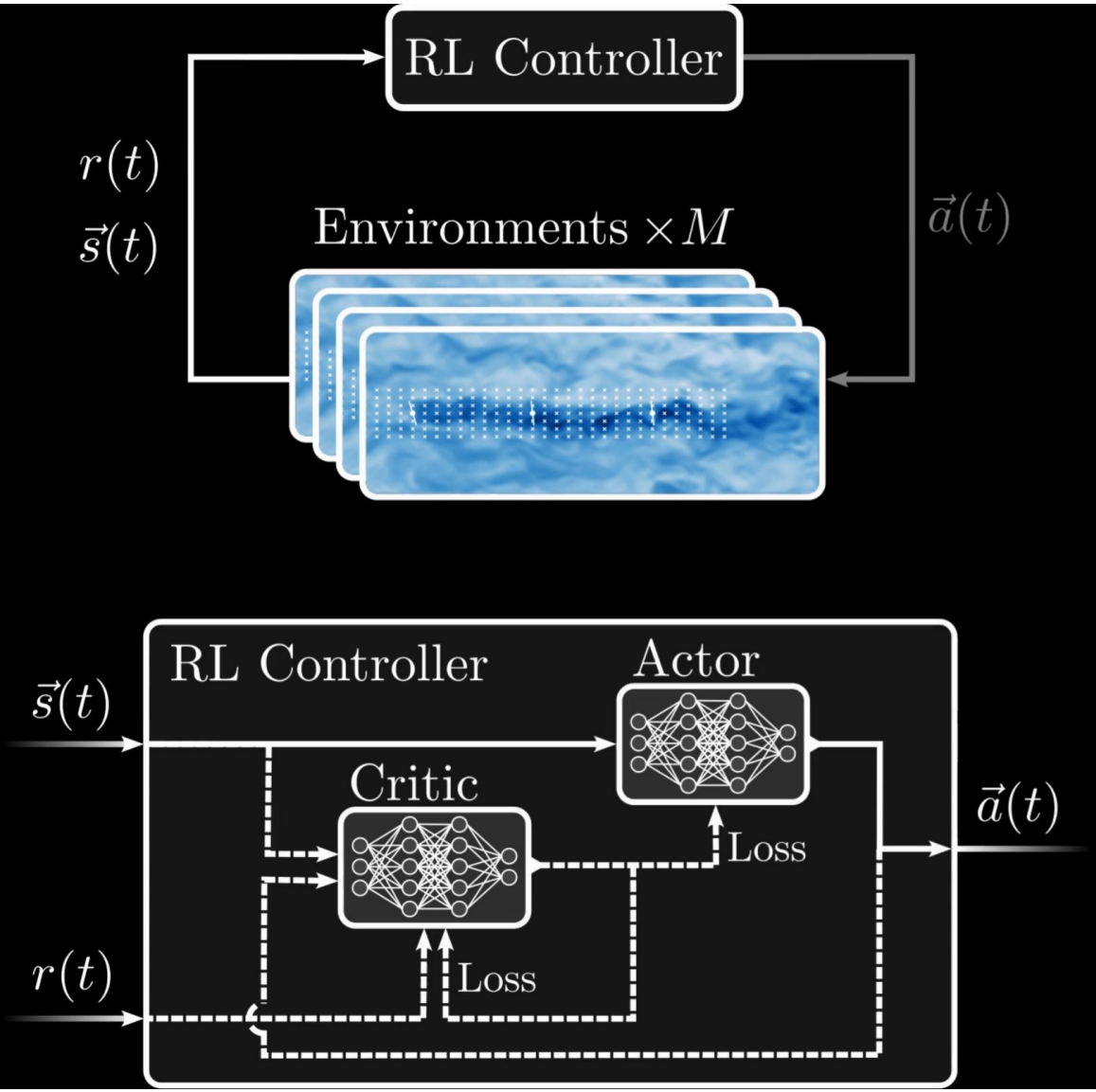
Experiment

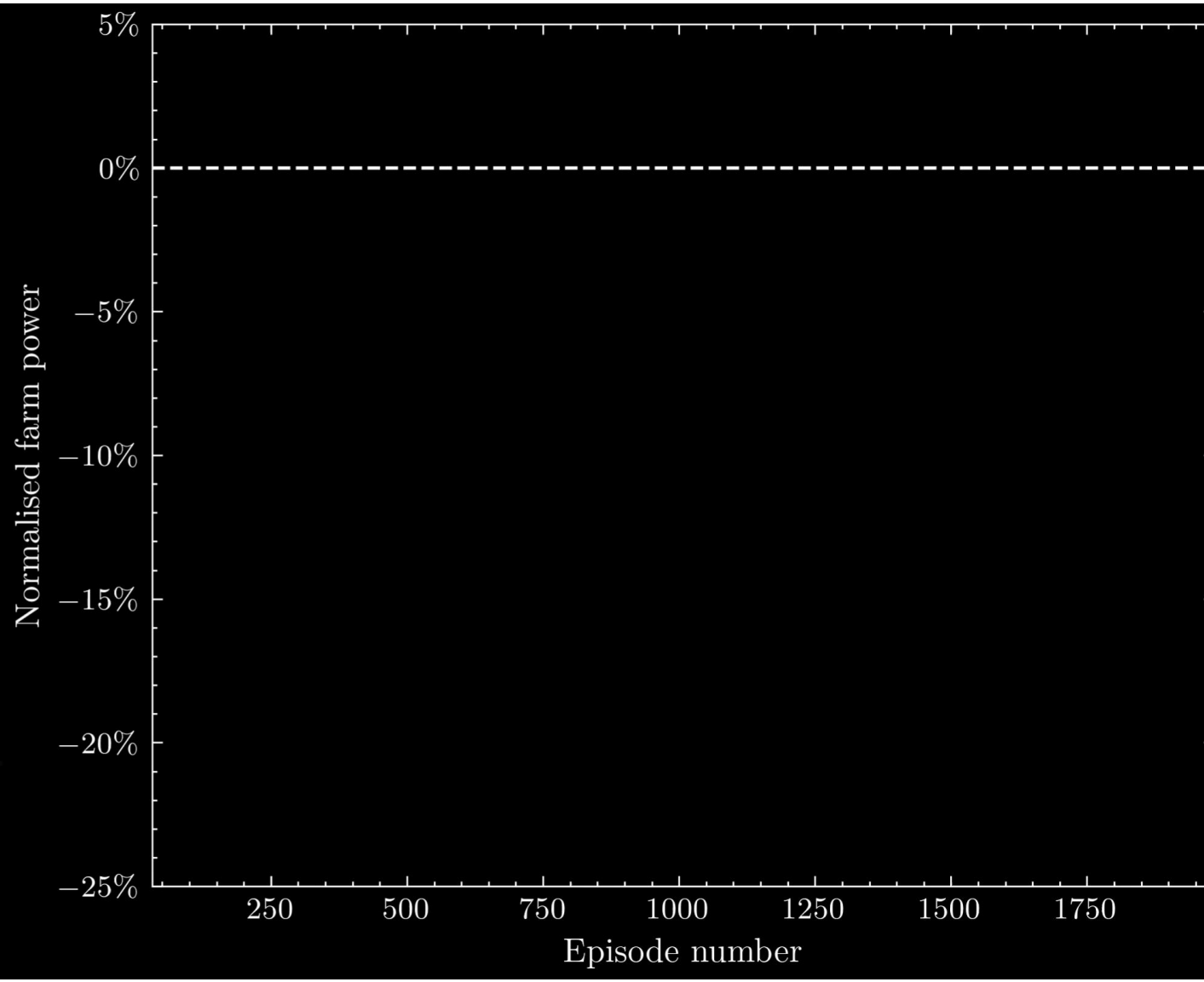
Setup on ARCHER2

- 32 Environments each with:
- 10 turbines
- 128 cores per environment (a node)
- 50 simulation timesteps per RL step
- Training every 500 steps

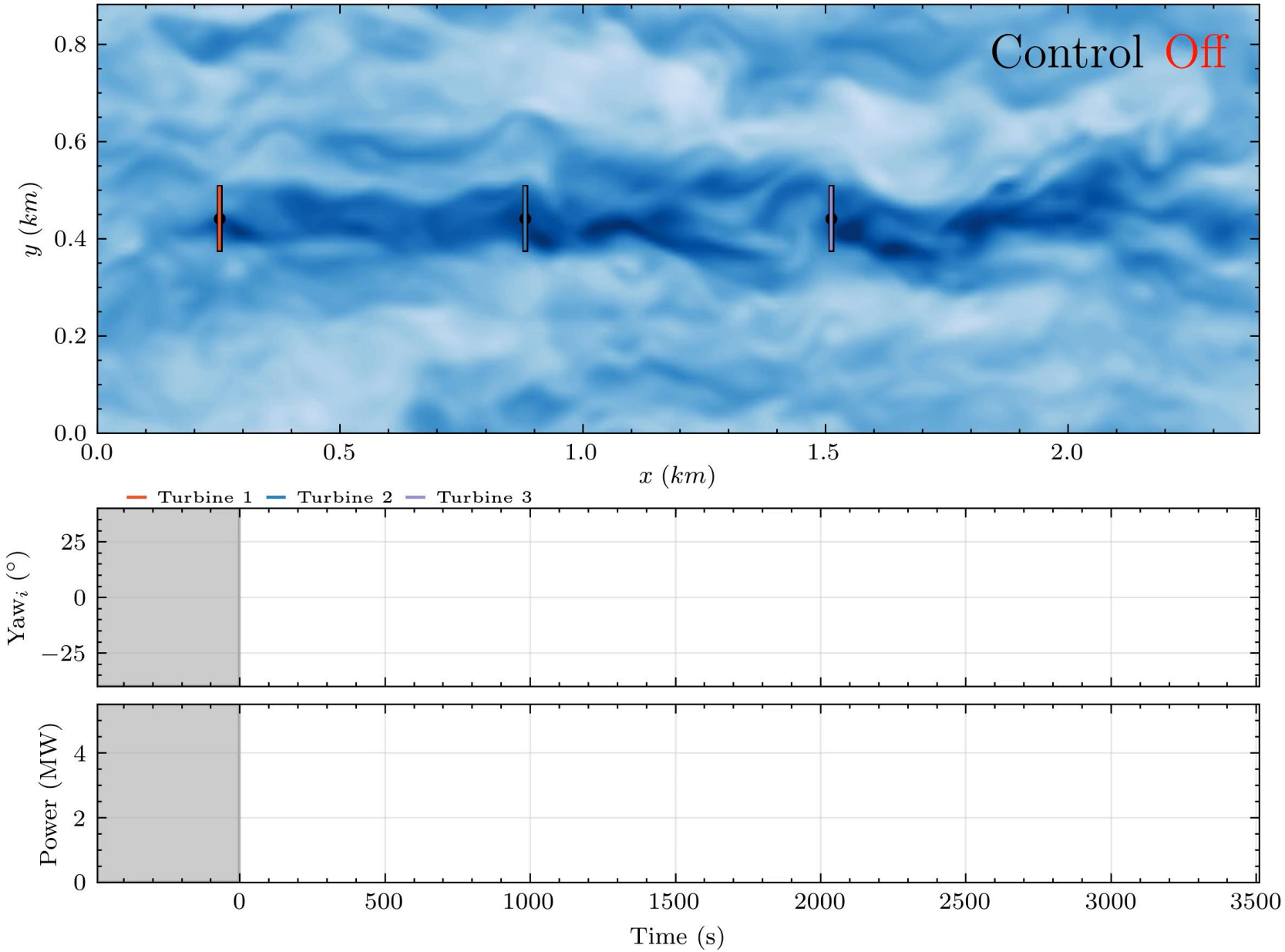


Training


RL Controller
$r(t)$
$\vec{s}(t)$
Environments $\times M$
$\vec{a}(t)$
RL Controller
Actor
$\vec{s}(t)$
Critic
$\vec{a}(t)$
Loss
$r(t)$
Loss


5%
0%
−5%
−10%
−15%
−20%
−25%
Normalised farm power
250
500
750
1000
1250
1500
1750
Episode number

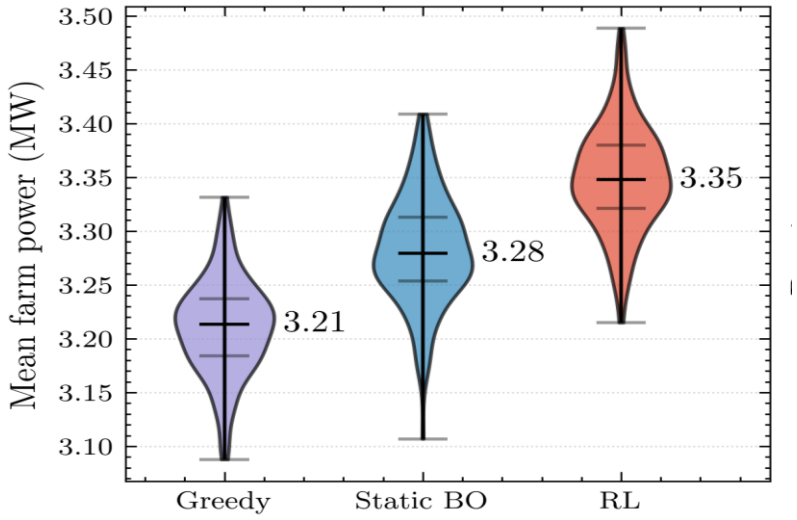
Evaluation

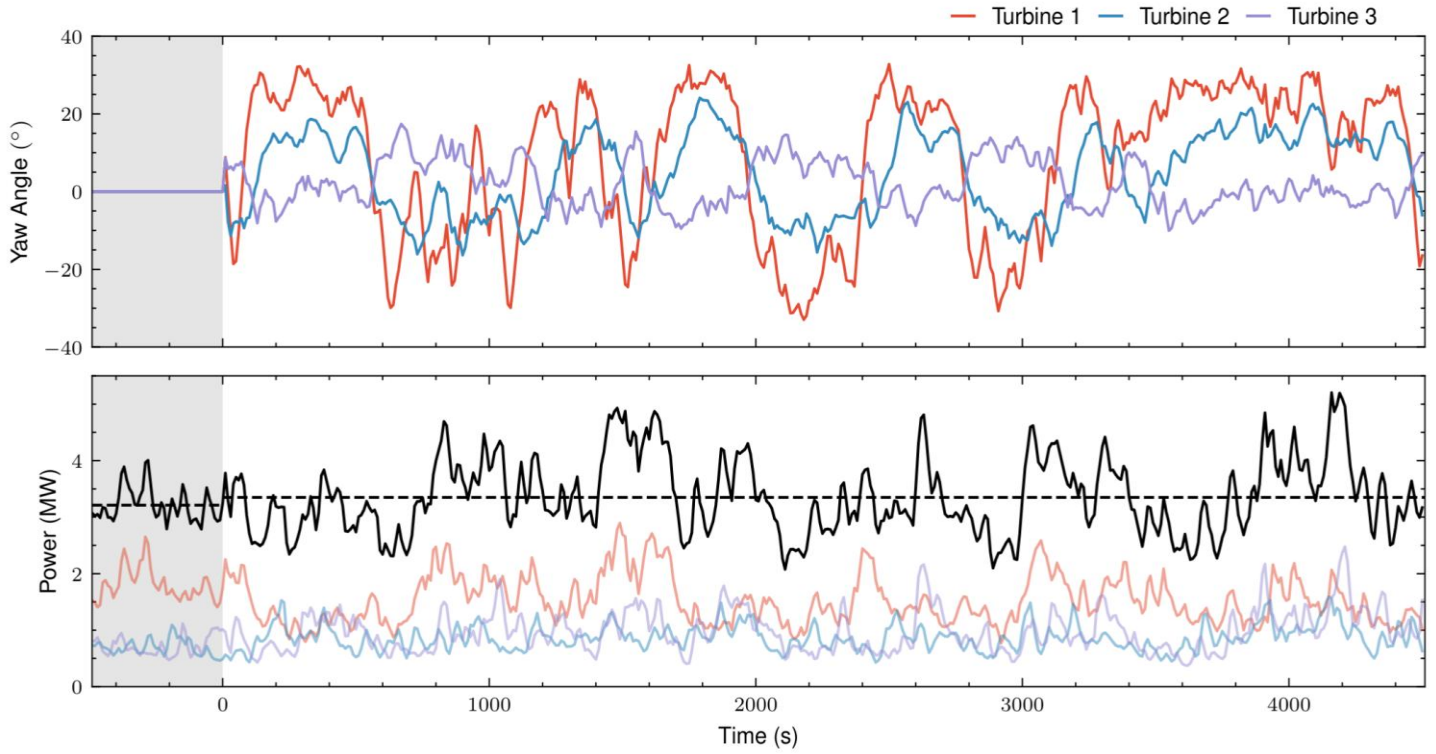

Control Off
0.8
0.6
0.4
0.2
0.0
y (km)
0.0
0.5
1.0
1.5
2.0
x (km)
Turbine 1
Turbine 2
Turbine 3
Yaw$_i$ (°)
25
0
−25
Power (MW)
4
2
0
0
500
1000
1500
2000
2500
3000
3500
Time (s)

Time series

After initial training the turbines settle down into being driven in sinusoidal pattern

Better power output than without steering.





Training a turbulence model using Relexi

Reinforcement learning in CFD

Modeling Assumption:

— Full Solution ~ Coarse Solution + Turbulence Model

Supervised Learning

— Inherent risk because ML model does not interact with the simulation

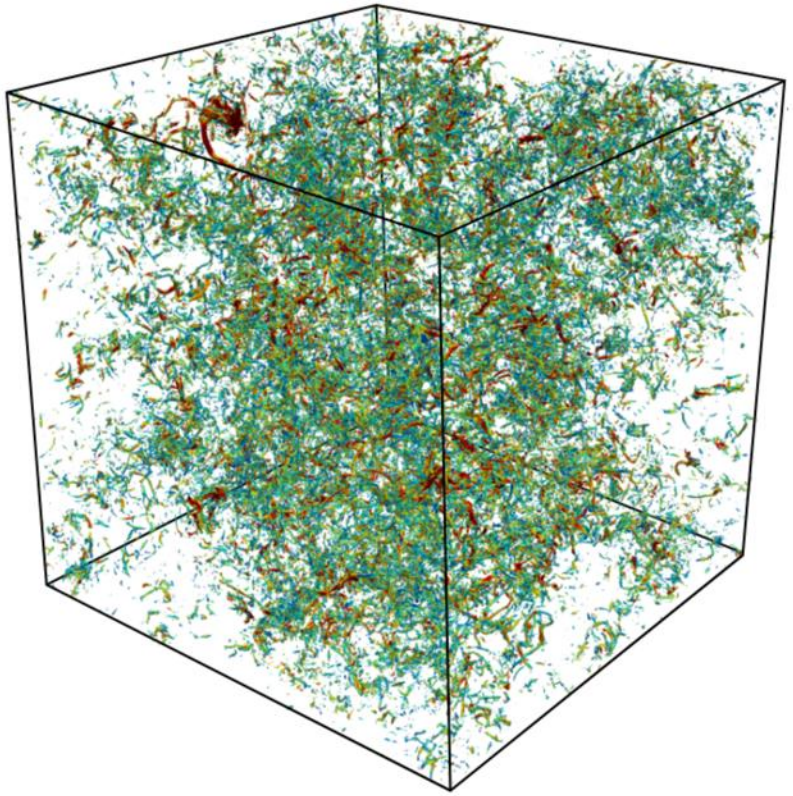
➔ instability, crashes, etc.

Reinforcement Learning solution:

— Train a model of turbulence in-situ to ensure stability

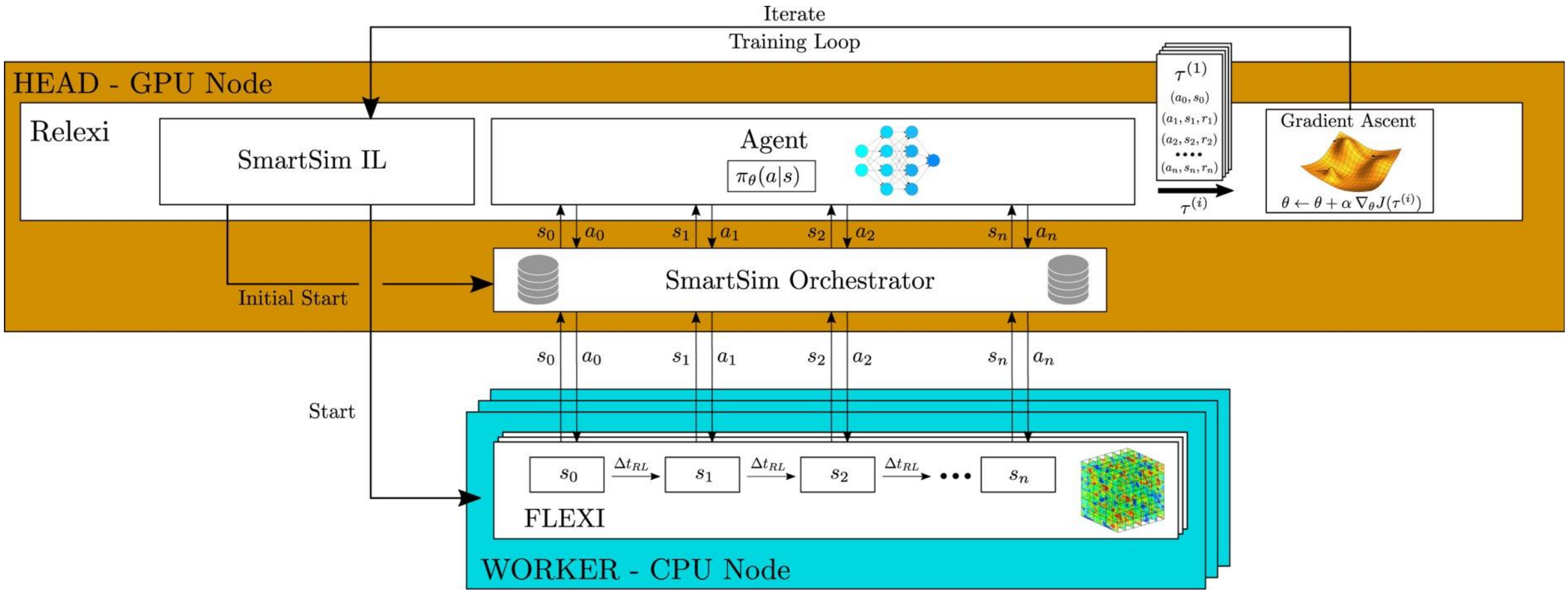
— Agent: predict Smagorinsky coefficient based on local u, v in LES

— Reward function: Compare LES+ML turbulent spectra to DNS solution

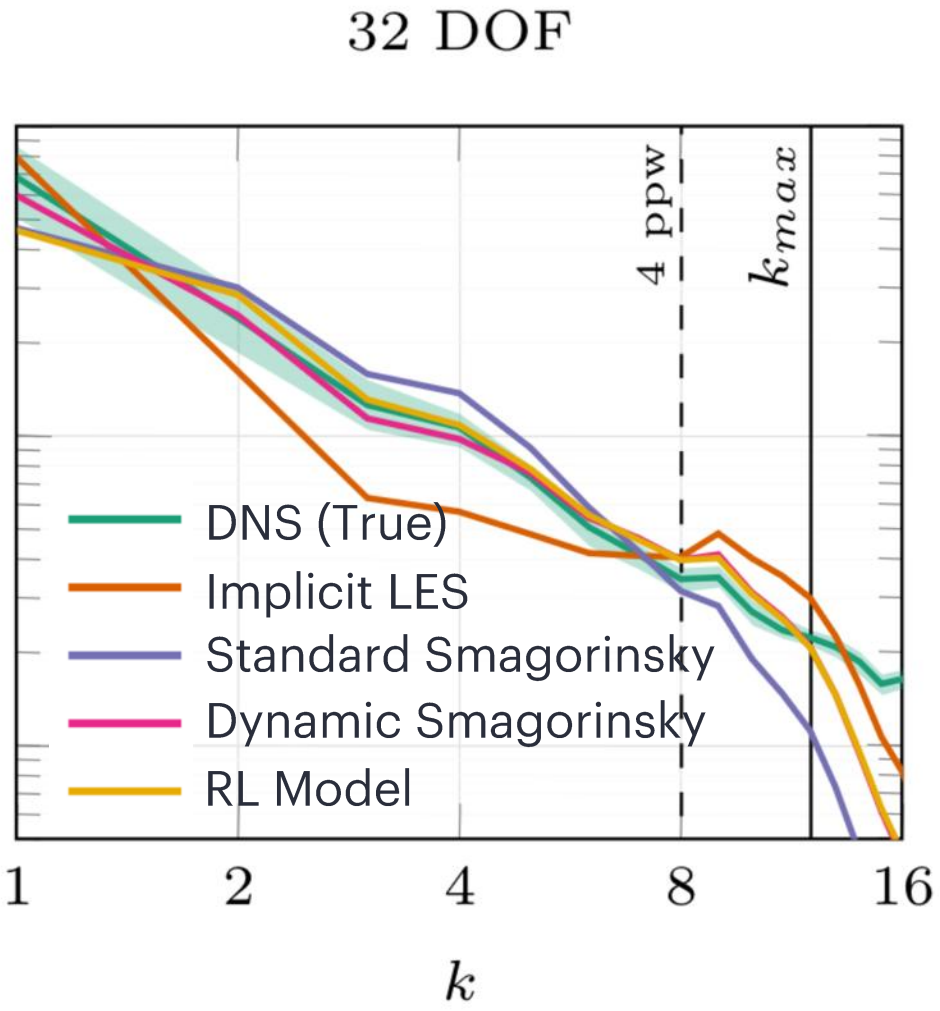


Work done by Kurz, Offenhäuser, and Beck [2022], https://doi.org/10.1016/j.ijheatfluidflow.2022.109094

Architecture of Relexi



RL Model reproduces “true” spectra



RL Model convergence takes ~1 day

—1,024 CPUs (8 nodes), 8 Nvidia A100s (1 node)

Closely reproduces energy spectra down to resolved wavenumbers

—Skill beyond even the ‘resolvable’ limit

—Some generalizability to different Reynolds numbers

Incurs about ~10% performance cost

—Small in comparison to Implicit LES

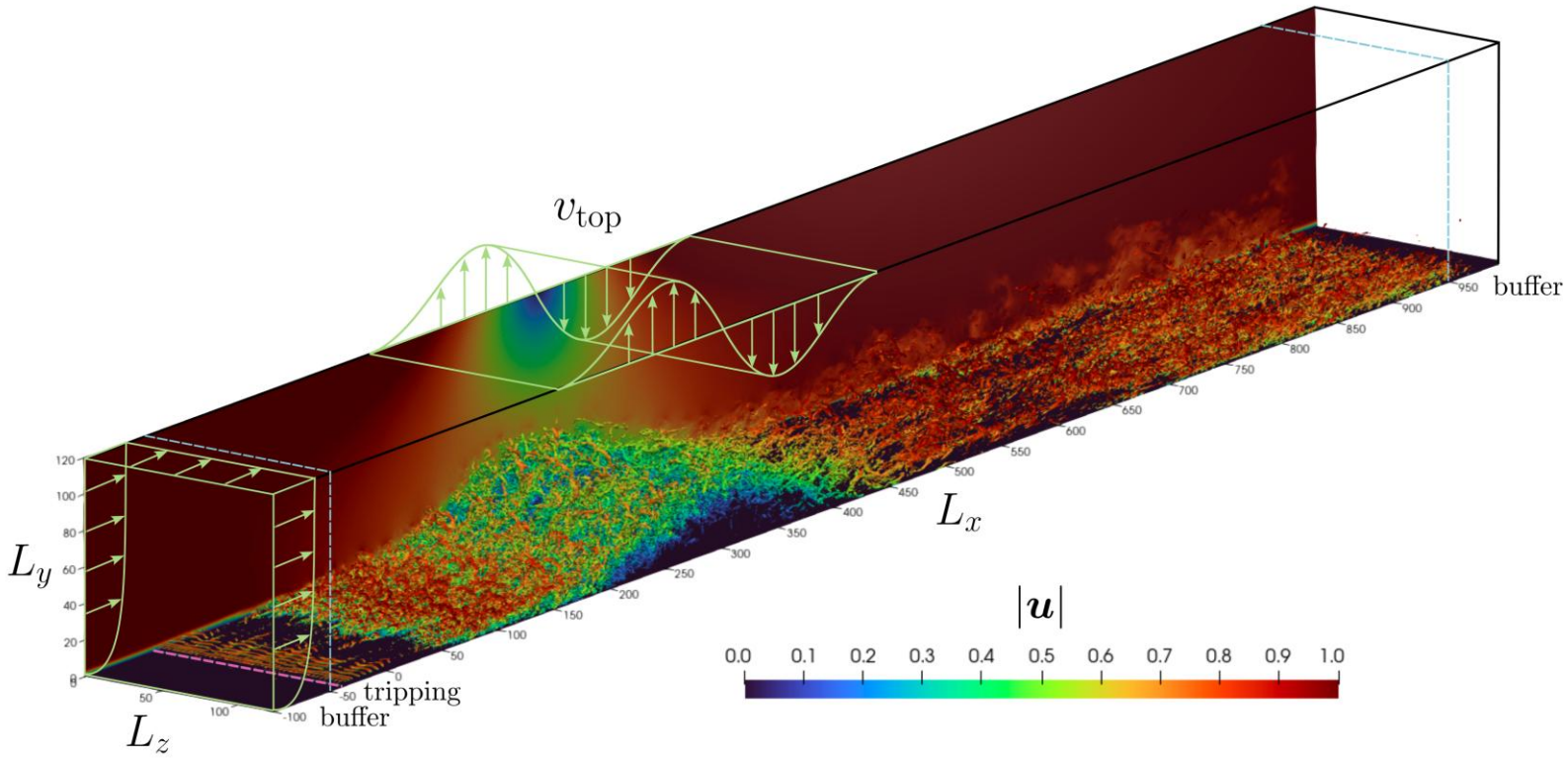
Flexibility in defining “reward” function allows different/multiple objectives

Article | Open access | Published: 07 February 2025

Deep reinforcement learning for active flow control in a turbulent separation bubble

Bernat Font ✉, Francisco Alcántara-Ávila, Jean Rabault, Ricardo Vinuesa ✉ & Oriol Lehmkuhl

Nature Communications



The big picture

Goal:
Reduce size of the bubble by moving actuators based discrete sampling of the flow

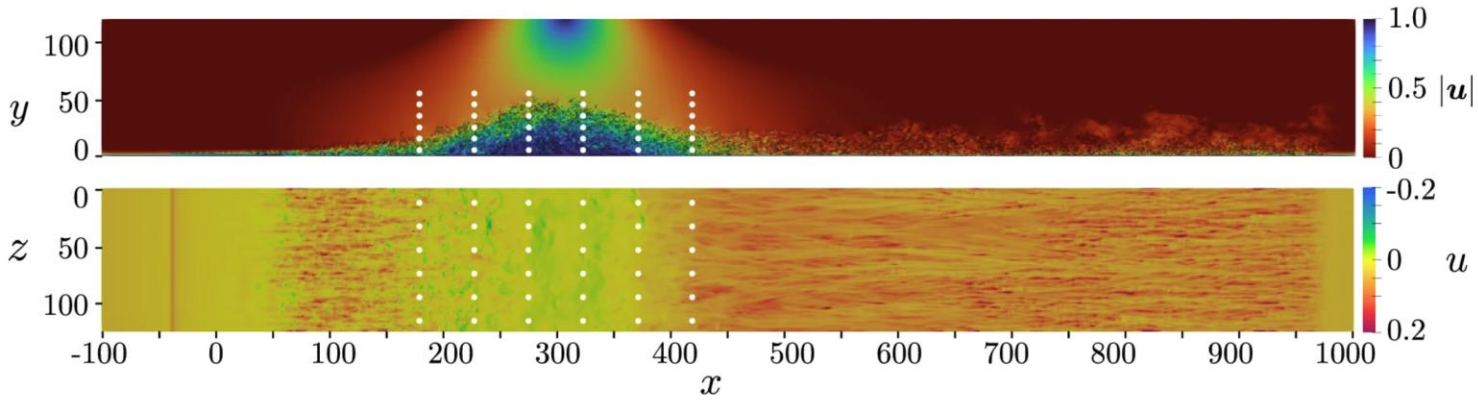
Method:
Deep Reinforcement Learning with small MLP to determine state of the actuators

Simulation:

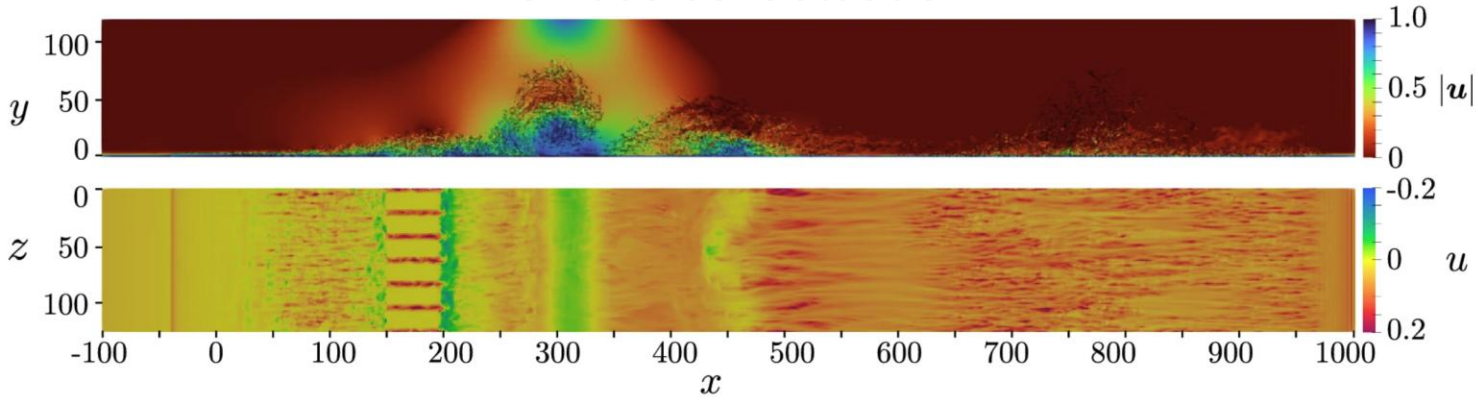
SOD2D: GPU-accelerated CFD code

Fully turbulent, high-Reynolds number flow

Non-Controlled Flow



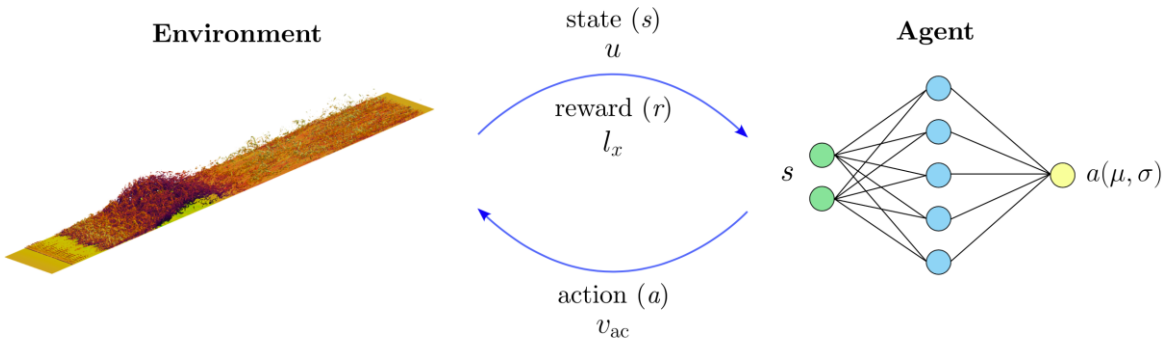
Sinusoidal actuation



Methods

Actuator control scheme

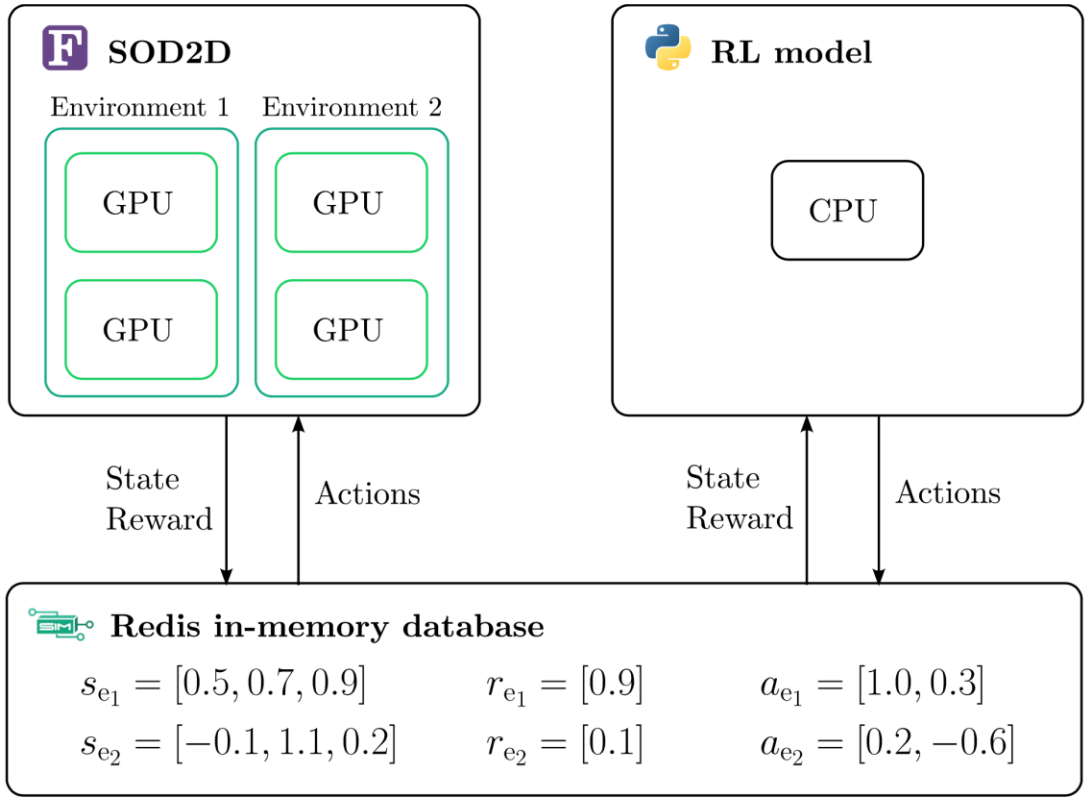
— Classical scheme: zero-net-mass-flux periodic control
  - Optimal, fixed-frequency sinuisoidal movement



— Method proposed: Deep Reinforcement Learning

— Reward increases as recirculation length of turbulent bubble decreases

— Training was performed using 24 parallel pseudo-environments running on 8 GPUs
  - 144 hours (6 days) in total (1152 GPU-hours)



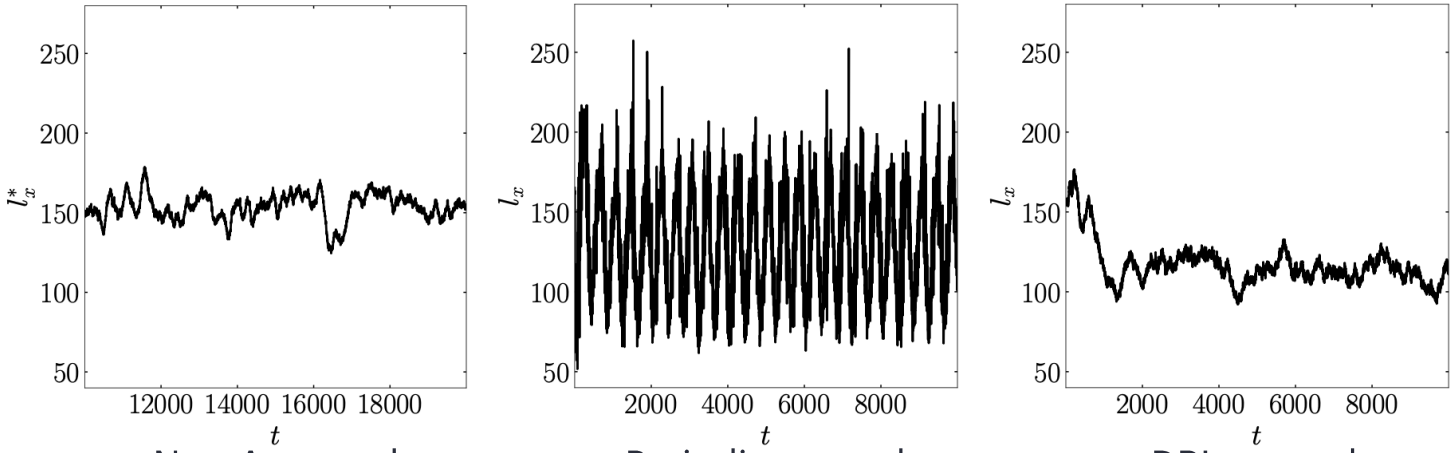
Results

DRL control improves reduction of TSB

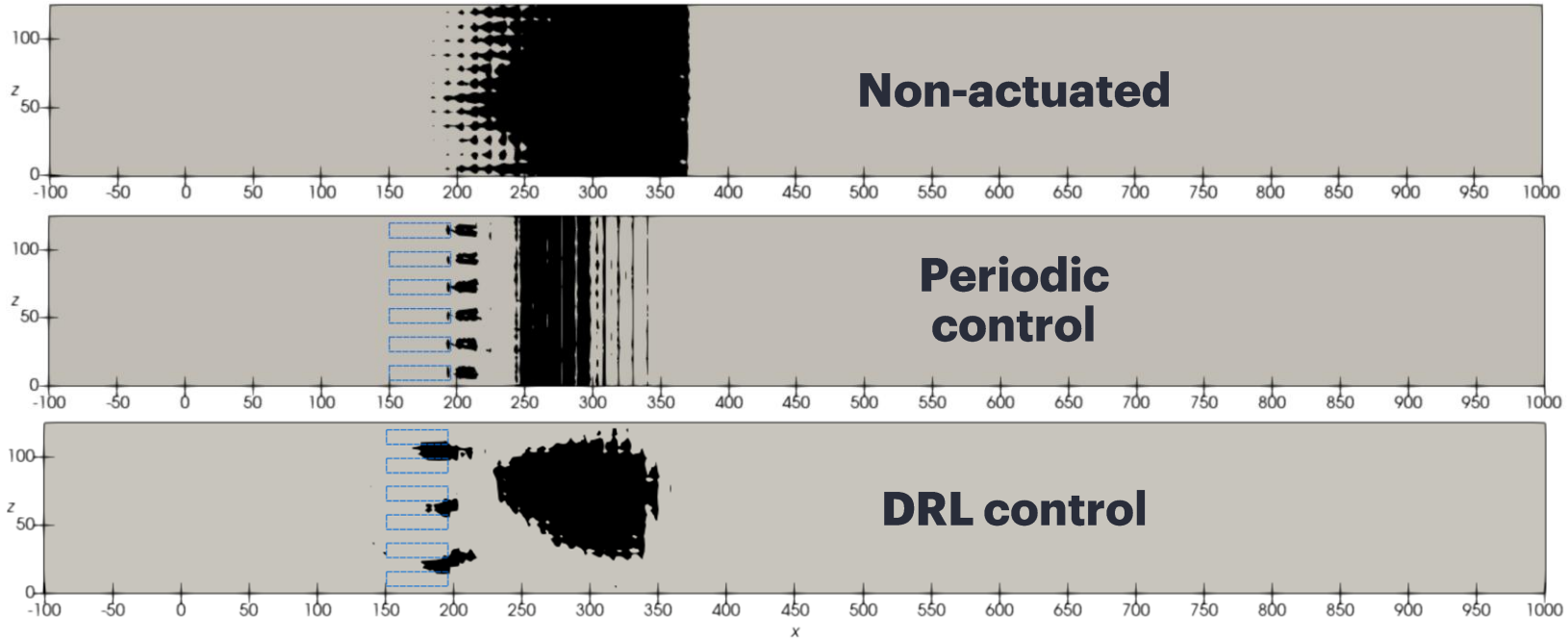
— 25.7% characteristic recirculation length reduction
  - Periodic control achieves 15.8%

— Less oscillations in recirculation length

Recirculation Length



Recirculation region



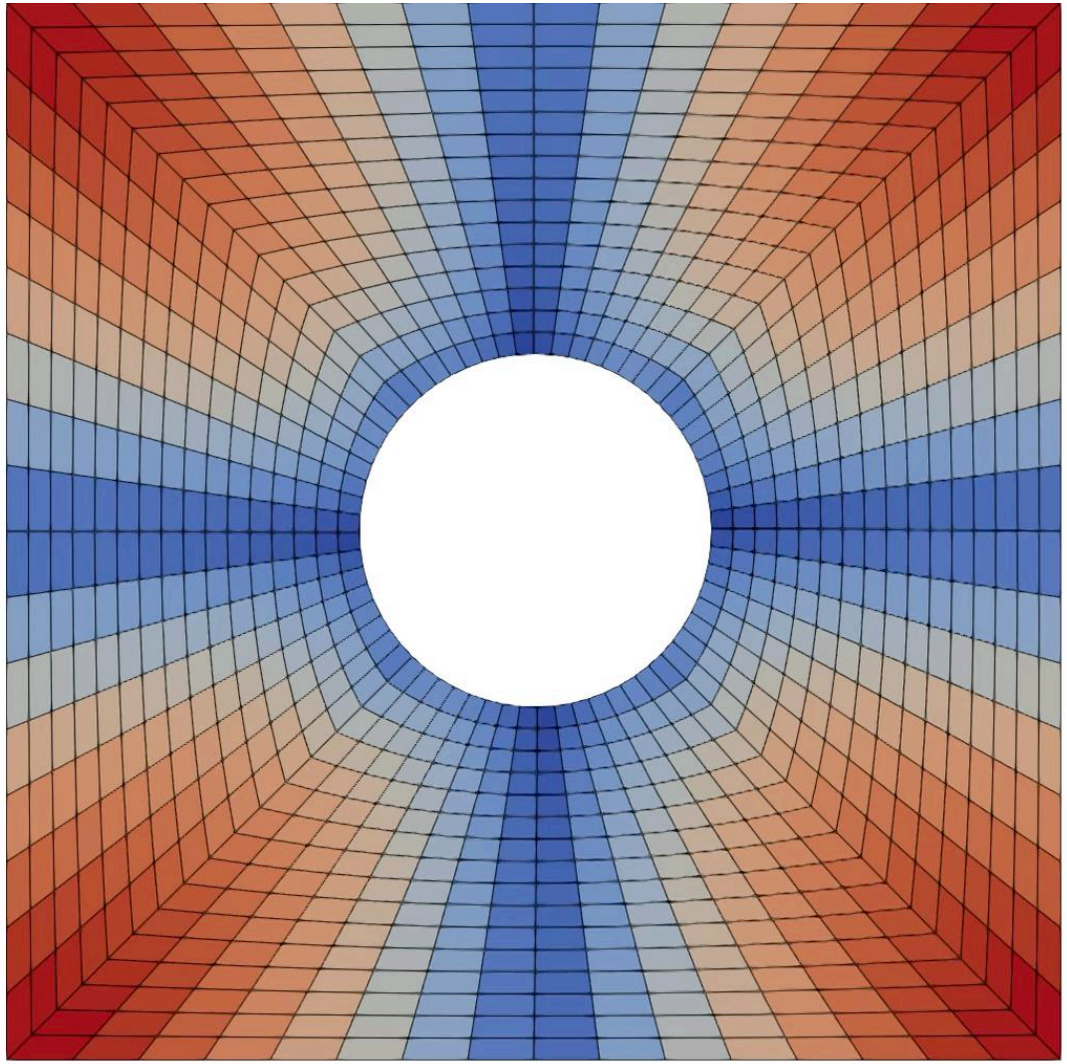
OpenFOAM Collaboration

OpenFOAM collaboration

SmartSim team and the OpenFOAM Data-Driven Modeling Special Interest Group have collaborated to create an OpenFOAM sub-module

Combining machine learning with computational fluid dynamics using OpenFOAM and SmartSim [Maric et al. 2025]

— SmartSim to cache streaming calculations
  - High-order statistics, PCA using distributed SVD
— Bayesian optimization
  - Use case: optimization of turbulence parameters
— Online training/inference for mesh motion
  - Can build physical constraints into the loss function when training



Sub-module API: three layers

1. **Service** API
   - High-level services for common workflows on OpenFOAM's geometric fields.
   - Includes methods like **`sendGeometricFields`**, which packs the field's internal data and sends it to the SmartRedis database.

2. **Developer** API
   - All methods from the developer layer do not interact directly with the SmartRedis database; instead, they handle a Dataset reference passed in as a first argument.

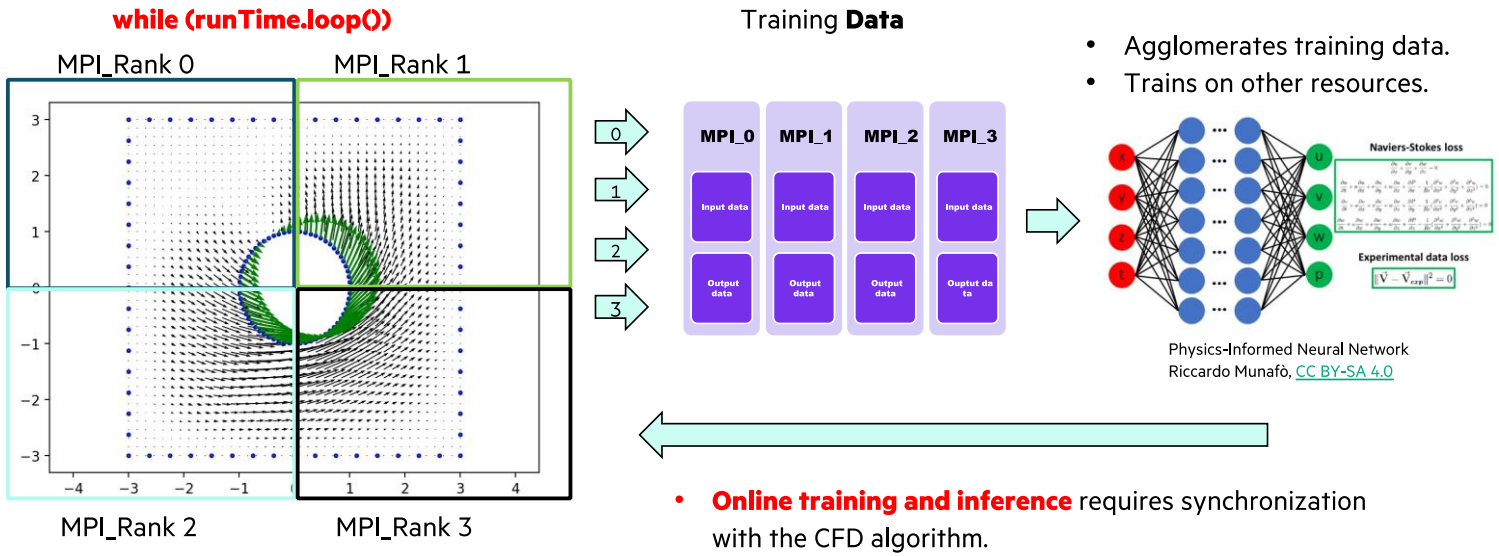
3. **Generic** API
   - facilitates generic interactions with the Database, which are intended as fallback methods
   - consistent approach to serialize and deserialize OpenFOAM List objects to and from SmartRedis tensors, maintaining relevant tensor dimensions, and storing them under a contiguous memory layout

```
struct smartRedisClient {
    //- Send a set of OpenFOAM fields of any type as SmartRedis tensors
    void sendGeometricFields (
        const wordList&,
        const wordList& patchNames = wordList{"internal"}
    );


    //- Send fields of type T to SmartRedis Dataset
    template<class T> void packFields (
        DataSet& ds,
        const wordList& fieldNames,
        const wordList& patchNames = wordList{"internal"}
    );


    //- Send a list of objects to SmartRedis DB
    template<class T>
    void sendList(const List<T>& lst, const word& listName);
};
```

ONLINE ML MESH MOTION



- Mesh motion problem: Given a moving boundary how should the interior part of the mesh move
  - Example: A rotating fan blade constantly changes the location of the blade
  - Example: Shape of the wing changes during landing/takeoff
- Traditional approach: Solve a boundary-condition problem
  - Laplace equation “diffuses” the motion of the boundary within the interior
  - Expensive as this has to be solved at every grid point
- AI/ML Approach: Use a small model (MLP) with physical constraints to learn the field
  - Inexpensive due to continual learning and subsampling
  - Model is trained to predict the boundary displacements based on *x*, *y* coordinates and satisfies physical constraints in the interior
  - Inference is then called to predict at every interior point

OpenFOAM sub-module recap

Allows OpenFOAM users to interact primarily with only OpenFOAM code

- No knowledge of SmartRedis is directly needed

Open source, available in this **public repo**: https://github.com/OFDataCommittee/openfoam-smartsim

Full explanation of module and examples available in open-access paper

- Combining machine learning with computational fluid dynamics using OpenFOAM and SmartSim

More advanced examples are currently in development

- Bayesian optimization for heat sink design
- 3d mesh motion using PDE constraints and/or metrics of mesh quality
  - This afternoon's last freeform activity

What's next for SmartSim? The name and more!

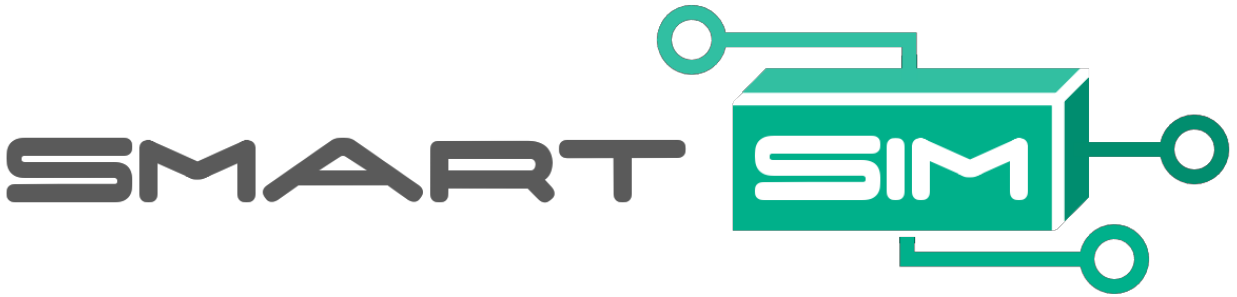
- HPE is partnering with Rutgers University and Princeton Plasma Physics Laboratory
  - **R**untime for **H**eterogeneous **AP**plications, **S**ervice **O**rchestration and **DY**namism (**RHAPSODY)**
  - New middleware for defining workflows for hybrid HPC/AI applications
  - https://github.com/radical-cybertools/rhapsody
  - Paper to be presented at International Supercomputing 2026: https://arxiv.org/pdf/2512.20795
- Why the change?
  - Broaden the userbase and combine development efforts
  - Remove reliance on proprietary, restrictive licenses
  - Improve performance and scalability by moving to HPC-first software solutions
- Question: "When should we expect the change to happen?"
  - Early demos are scheduled for summer 2026
  - Capability parity with SmartSim/SmartRedis by end of 2026
- Question: "But wait, why should I write SmartSim/SmartRedis code?"
  - Gain experience writing loosely-coupled, data-driven workflows
  - When ready, we will help you port your workflows over to RHAPSODY

Further reading

Where can I find more?

Other SmartSim Information

- Contact: CrayLabs@hpe.com
- SmartSim Repository: https://github.com/CrayLabs/SmartSim
  - Includes docker container with tutorials!
- SmartRedis Repository: https://github.com/CrayLabs/SmartRedis
- SmartSim Case studies: https://github.com/CrayLabs/SmartSim-Zoo
- SmartSim Slack workspace (link on repo)



Documentation

www.craylabs.org

SmartSim 0.8.0 documentation

Versions

Getting Started

- Introduction
- Basic Installation
- Installation on specific platforms
- Contributing Guide
- Contributing Examples

Tutorials

- Getting Started
- Online Analysis
- Online Inference
- Online Training

SmartSim

- Experiments
- Run Settings
- Batch Settings
- Model
- Ensemble
- Orchestrator
- Logger
- ML Features
- Dragon
- SmartSim API

Contents

- Library Design

light/dark

Introduction

SmartSim enables scientists to utilize machine learning inside traditional HPC workloads

SmartSim provides this capability by

1. Automating the deployment of HPC workloads and distributed, in-memory storage (Redis).
2. Making TensorFlow, Pytorch, and ONNX callable from Fortran, C, and C++ simulations.
3. Providing flexible data communication and formats for hierarchical data, enabling online analysis, visualization, and processing of simulation data.

The main goal of SmartSim is to provide scientists a flexible, easy to use method for interacting at runtime with the data generated by simulation. The type of interaction is completely up to the user.

- Embed calls to machine learning models inside a simulation
- Create hooks to manually or programmatically steer a simulation
- Visualize the progression of a simulation integration from a Jupyter notebook

The figure below shows the architecture of SmartSim for a given use case. SmartSim can create, configure and launch workloads (called a `Model`), as well as groups of workloads (`Ensembles`). The data communication between a workload and in-memory storage is handled by the SmartRedis clients, available

GETTING STARTED

- Introduction
- Installation
- Community
- Contributing Examples

TUTORIALS

- Getting Started
- Online Analysis
- Online Inference
- Online Training
- Ray Integration

SMARTSIM

- Experiments
- Orchestrator
- Launchers
- SmartSim API

SMARTREDIS

- SmartRedis
- Integrating into a Simulation
- Python
- C++
- Fortran
- Data Structures
- Runtime Requirements
- SmartRedis API

REFERENCE

Contents

- Experiment
- Settings
- Orchestrator
- Model
- Ensemble
- Machine Learning
- Slurm
- Ray

SmartSim API

Experiment

`Experiment.__init__`(name[, exp_path, launcher])	Initialize an Experiment instance
`Experiment.start`(*args[, block, summary, ...])	Start passed instances using Experiment launcher
`Experiment.stop`(*args)	Stop specific instances launched by this `Experiment`
`Experiment.create_ensemble`(name[, params, ...])	Create an `Ensemble` of `Model` instances
`Experiment.create_model`(name, run_settings)	Create a general purpose `Model`
`Experiment.create_database`([port, db_nodes, ...])	Initialize an Orchestrator database
`Experiment.create_run_settings`(exe[, ...])	Create a `RunSettings` instance.
`Experiment.create_batch_settings`([nodes, ...])	Create a `BatchSettings` instance
`Experiment.generate`(*args[, tag, overwrite])	Generate the file structure for an `Experiment`
`Experiment.poll`([interval, verbose, ...])	Monitor jobs through logging to stdout.
`Experiment.finished`(entity)	Query if a job has completed.
`Experiment.get_status`(*args)	Query the status of launched instances
`Experiment.reconnect_orchestrator`(checkpoint)	Reconnect to a running `Orchestrator`

Thank You

© 2026 Hewlett Packard Enterprise Development LP