Hack the Turbulence:

Enhancing Flow Resolution with AI

Hesam Tofighian
Karlsruhe Institute of Technology (KIT) - KCDS

16. - 19. September 2025

Abstract. Turbulence spans a vast range of interacting scales. Resolving all dynamically relevant eddies with DNS is expensive, whereas coarse simulations are fast but miss crucial small-scale physics. In this hackathon you will develop deep-learning models that map low-resolution turbulent velocity fields to high-resolution reconstructions *while respecting the underlying physics*. You will work with homogeneous isotropic turbulence (HIT) box data, receive a baseline PyTorch implementation (single-GPU/CPU and multi-GPU DDP) code, and use KIT's compute resources.

Scientific Background: How Turbulence Emerges

Turbulent motion is governed by the incompressible Navier–Stokes (NS) equations with forcing [2]:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F}, \qquad \nabla \cdot \mathbf{u} = 0, \tag{1}$$

where $\mathbf{u} = (u, v, w)$ is velocity, p is pressure, ν is kinematic viscosity, and $\mathbf{F}$ is an external body force (e.g. large-scale stirring sustaining HIT).

Inertia vs. diffusion. The nonlinear advection $(\mathbf{u}\cdot\nabla)\mathbf{u}$ redistributes momentum and transfers energy across scales; the viscous term $\nu\nabla^2\mathbf{u}$ dissipates it. Their competition is summarized by the Reynolds number $Re = UL/\nu$. High Re implies advection-dominated dynamics, instability, and ultimately turbulence. Let $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ denote vorticity. If we visualize iso-surfaces of the vorticity magnitude $|\boldsymbol{\omega}|$ in a turbulent flow, we observe multi-scale, tube-like coherent structures concentrated in regions of intense rotation; see Fig. 3.

Taking the curl of Eq. (1) yields

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} + \nabla \times \mathbf{F}. \tag{2}$$

The *vortex-stretching* term $(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$ (present in 3D) amplifies $|\boldsymbol{\omega}|$ when a vortex tube is elongated, sharpening gradients and feeding smaller scales. Diffusion $\nu\nabla^2\boldsymbol{\omega}$ smooths these gradients; stretching intensifies until diffusion becomes comparable, or the structure becomes unstable and fragments into smaller eddies. This stretch $\rightarrow$ fold $\rightarrow$ reconnect/cancel cycle underpins the forward cascade from large to small scales; see Fig. 2. The cascade continues down to the Kolmogorov length scale [2]. Capturing all motions down to Kolmogorov length scale is prohibitively expensive in simulations, which motivates a data-driven super-resolution approach [4]: reconstruct the missing small-scale structures from coarse fields.

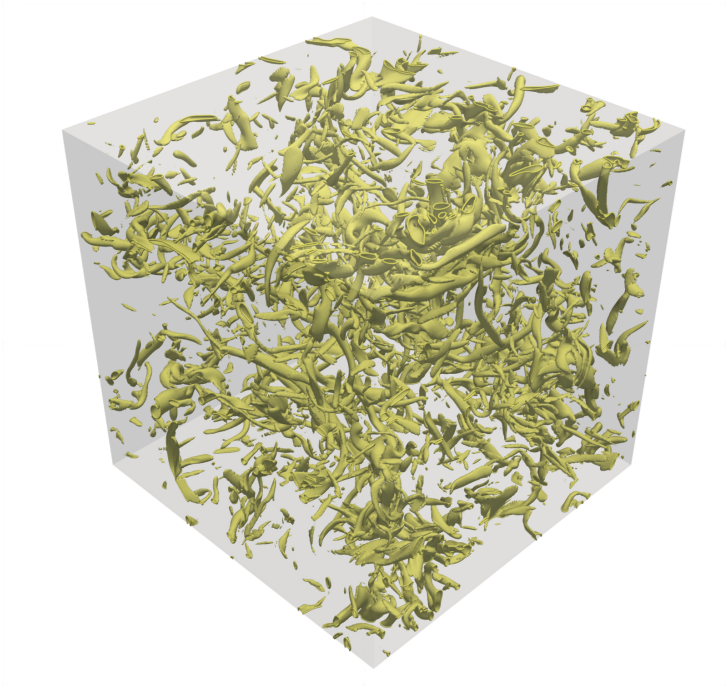

Figure 1: Iso-surfaces of vorticity magnitude $|\boldsymbol{\omega}|$ in homogeneous isotropic turbulence (HIT).

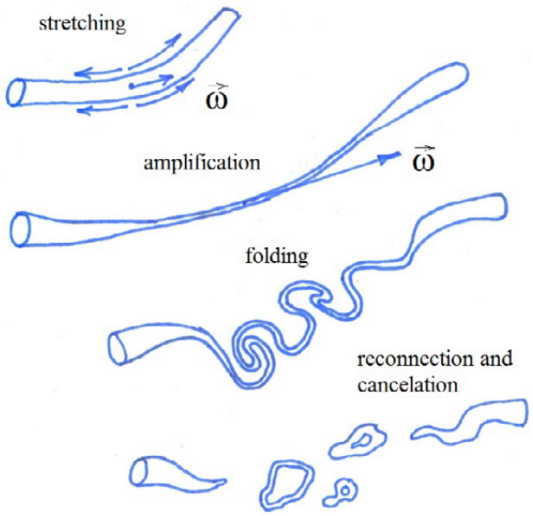

Figure 2: Schematic of vortex-tube dynamics: stretching and amplification, folding, and reconnection/cancellation, driving a cascade of smaller structures [1].

Hackathon Task

Goal. Learn a mapping $\mathcal{G}$: LR $\rightarrow$ HR that reconstructs high-resolution velocity fields from filtered/downsampled inputs while remaining faithful to physics and turbulence statistics.

You will receive:

- **HIT dataset [3]:** 3D velocity fields (u, v, w) stored as NumPy arrays; high-resolution samples and corresponding low-resolution inputs obtained by filtering and down-sampling.
- **Baseline code:** a PyTorch 3D CNN generator with residual-in-residual dense blocks (RRDB) [5], plus data augmentation and physics-aware losses.

What you can improve (suggestions). Try U-Nets, attention blocks, physics-informed losses, or matching spectra, adversarial training, Fourier/Neural Operator architectures, Diffusion models, Kolmogorov arnold neural network, etc.

Dataset and Format

HIT boxes. The dataset consists of periodic 3D cubes representing homogeneous isotropic turbulence (HIT) with velocity components (u, v, w) [3]. Each full-resolution snapshot has size 1024^3 with three channels corresponding to the velocity components, stored in the shape `[1024, 1024, 1024, 3]`.

Sub-box extraction. To alleviate memory constraints, each large cube has already been partitioned into smaller sub-boxes of size 72^3. Participants can directly use these pre-split sub-boxes or employ the provided script `makeSubBoxes.py` to generate sub-boxes of arbitrary size for training.

Evaluation

Participants will be provided with 3D low-resolution HIT snapshots and tasked with reconstructing high-resolution velocity fields at **4× resolution in each spatial direction** with their model. This means, for example, that a coarse cube of size 50^3 must be super-resolved to 200^3.

Models will be compared by a mix of accuracy and physics:

- **Reconstruction:** Mean squared error (MSE); gradient error.
- **Physics:** Divergence norm, extreme events in vorticity/gradient statistics, energy spectra recovery.
- **Computation:** Computational efficiency during inference.

HPC: Running on HAICORE@KIT (Helmholtz AI)

Registration. All members of Helmholtz Association institutions can self-register for HAICORE@KIT. Use the online form: haicore/registration.

Login (SSH). After your account is approved, log in via:

```
ssh username@haicore.scc.kit.edu
```

Workspace management. Allocate a project workspace (persistent project storage):

```
ws_allocate name_of_workSpace
```

List/recall your workspaces:

```
ws_list
```

Copy code/data. From your local machine to your HAICORE workspace:

```
scp -r data_directory username@haicore.scc.kit.edu:/work_space_directory
```

Python environment. Create and activate a virtual environment (and optionally add activation to `/.bashrc`):

```
python3 -m venv kcdsEnv
source kcdsEnv/bin/activate
# (optional) echo "source ~/kcdsEnv/bin/activate" >> ~/.bashrc
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
pip install numpy matplotlib
```

Preparing HIT sub-boxes. The raw snapshots are $1024^3 \times 3$ (velocity components). Working directly at full resolution exceeds typical GPU memory. The snippet below circularly pads by 4 cells on each face (for a 9^3 smoothing filter) and extracts 64^3 interior sub-boxes, yielding 72^3 padded tiles.

```
import numpy as np
import torch
import torch.nn.functional as F
import os

# Load a snapshot shaped [1024, 1024, 1024, 3] (channel-last)
velocity = np.load('U_T1.npy')                        # shape: [D, H, W, 3]
velocity = torch.from_numpy(velocity).permute(3,0,1,2)  # -> [3, D, H, W]
velocity = velocity.unsqueeze(0)                          # -> [1, 3, D, H, W]

# Circular pad by 4 on each side of D, H, W -> [1, 3, 1032, 1032, 1032]
velocity = F.pad(velocity, (4,4, 4,4, 4,4), mode='circular')

# Convert back to NumPy for saving
velocity = velocity.numpy()

# Make output directory
output_dir = "./sub_boxes"
os.makedirs(output_dir, exist_ok=True)

# Extract 64^3 interior per tile, stored with 4-cell pad on each face (72^3 total)
for i in range(0, 1024, 64):
    for j in range(0, 1024, 64):
        for k in range(0, 1024, 64):
            sub_box = velocity[:, :, i:i+72, j:j+72, k:k+72]  # [1, 3, 72, 72, 72]
            fname = f"U_JH_T1_{i}_{j}_{k}.npy"
            np.save(os.path.join(output_dir, fname), sub_box)
```

You can adjust the interior size and padding to match your filter width.

Submitting training jobs (SLURM). Create `run_train.sh`:

```
#!/bin/bash
#SBATCH --account=kcds
#SBATCH --partition=normal
#SBATCH --job-name=myTraining
#SBATCH --ntasks=1
#SBATCH --gres=gpu:full:2
```

```
#SBATCH --time=01:00:00
#SBATCH --mem=64G
#SBATCH --cpus-per-task=8
#SBATCH --mail-type=ALL
##SBATCH --mail-user=your_email@domain
#SBATCH --output=training_%j.log
#SBATCH --error=training_%j.err

module purge
export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
source ~/kcdsEnv/bin/activate
python train_DataParallel.py
```

Submit and monitor:

```
sbatch run_train.sh
```

Inference. After training, run:

```
python inference.py
```

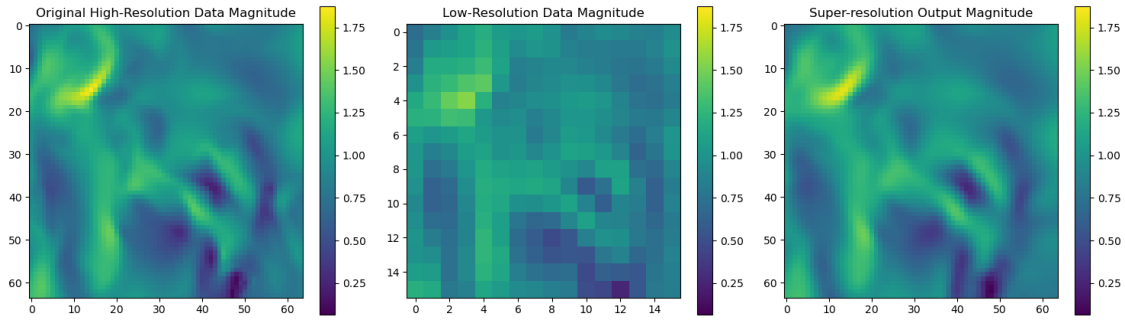

Figure 3: Output of the inference.py.

References

[1] I Cherunova and N Kornev. Lectures on computational fluid dynamics. *bookboon. com*, 2015.

[2] Stephen B Pope. Turbulent flows. *Measurement Science and Technology*, 12(11):2020–2021, 2001.

[3] FORCED ISOTROPIC TURBULENCE DATA SET. The johns hopkins turbulence databases (jhtdb).

[4] Filippos Sofos and Dimitris Drikakis. A review of deep learning for super-resolution in fluid flows. *Physics of Fluids*, 37(4), 2025.

[5] Hesam Tofighian, Jordan A Denev, and Nikolai Kornev. A conditional deep learning model for super-resolution reconstruction of small-scale turbulent structures in particle-laden flows. *Physics of Fluids*, 36(11), 2024.