

Cold Plate CFD and AI Emulation Data Readme

Description:

This dataset accompanies Curl & Hu (2026). It contains 2000 steady-state CFD simulations of a liquid-cooled, dual-chip, pin-fin cold plate run in ANSYS Fluent, the POD-NN surrogate trained on those simulations, and the source code that reproduces every figure in the paper from the published data.

The DOE varies six boundary conditions (chip 1 / chip 2 surface heat fluxes, inlet 1 / inlet 2 mass flow rates, and inlet 1 / inlet 2 total temperatures) across the operational space of the cold plate using Latin Hypercube Sampling. The CFD model solves the steady, incompressible Navier-Stokes equations coupled with the energy equation and Menter's $k-\omega$ SST turbulence model on a $\sim 50,000$ -element 3D grid; only data along the central yz mid-plane (11,110 nodes) is used in the published surrogate.

Headline results from the paper:

- POD with $r = 4$ modes captures 99.32% of the temperature-field variance.
- The trained POD-NN predicts a full 2D temperature field $412,500\times$ faster than the underlying CFD solve.
- A 100-sample training set is sufficient to reach the error floor; the iteration-cost crossover with CFD occurs at 100.5 design iterations (~ 2.76 hrs).

Data structure:

The dataset files can be read as follows:

```
/dataset
├── 1_Dataset/                                # CFD simulation outputs and DOE configuration
│   ├── sim_0001.npz ... sim_2000.npz        # 2000 NumPy archives, one per simulation
│   └── model_setup.json                      # DOE configuration (boundary-condition values)
├── 2_AnalyzedData/                          # Trained surrogate and derived results
│   ├── trained_model/                      # Canonical POD-NN referenced in the paper
│   │   ├── pod_nn.h5                      # Keras / TensorFlow network weights
│   │   ├── pca.pkl                        # sklearn PCA: 4 POD modes + snapshot mean
│   │   ├── param_scaler.pkl               # StandardScaler for the 6 input parameters
│   │   ├── mode_scaler.pkl               # StandardScaler for the 4 POD coefficients
│   │   └── training_history.json           # Per-epoch train / validation MSE
│   └── dataset_size_sensitivity.csv         # Raw numbers behind the sensitivity plots
├── 3_SourceCode/                            # Scripts reproducing every figure in the paper
│   ├── _utils.py                          # Shared module; all hyperparameters live here
│   ├── 01_train_model.py ... 12_rth_latent_space.py
│   ├── requirements.txt                    # Minimum dependency list
└── README                                  # This file
```

Files and variables:

The 1_Dataset folder contains:

sim_0001.npz through sim_2000.npz: 2000 NumPy compressed archives, one per CFD simulation. The i -th file corresponds to the i -th boundary-condition row in model_setup.json. Open with `numpy.load(path, allow_pickle=True)`. Each archive contains 14 keys, all stored as float arrays:

- "yz-mid|temperature" — (11110,) temperature (K) on the central yz mid-plane (the field used in the paper)

- "yz-mid|coordinates" — (11110, 3) X/Y/Z node coordinates (m) for the yz mid-plane
- "zx-mid|temperature" — (40939,) temperature (K) on the zx mid-plane
- "zx-mid|coordinates" — (40939, 3) X/Y/Z node coordinates (m) for the zx mid-plane
- "bottom|temperature" — (3193,) temperature (K) on the bottom-face cooling boundary
- "bottom|coordinates" — (3193, 3) X/Y/Z node coordinates (m) for the bottom face
- Scalar outputs, each shape (1,): "chip1_tavg|temperature", "chip1_tmax|temperature", "chip2_tavg|temperature", "chip2_tmax|temperature", "outlet_tavg|temperature", "outlet_tmax|temperature", "pdrop_1|temperature", "pdrop_2|temperature"

`model_setup.json`: DOE configuration produced by the original ANSYS / PyFluent workflow. Required by every analysis script in `3_SourceCode/` — they read it to reconstruct the per-sample input parameter values. Top-level keys:

- "timestamp" — when the DOE was generated.
- "model_inputs" / "model_outputs" — metadata lists describing the parameter and output channels.
- "doe_configuration" — nested dict {boundary_name: {param_name: [value_for_sample_1, ..., value_for_sample_2000]}}. The four boundary names are chip1, chip2, inlet1, inlet2; their parameters are chip1.thermal.heat_flux, chip2.thermal.heat_flux, inlet1.momentum.mass_flow_rate, inlet1.thermal.total_temperature, inlet2.momentum.mass_flow_rate, inlet2.thermal.total_temperature.
- "case_file" — local path to the original ANSYS .cas.h5 file (informational only; the case file itself is not redistributed here).

The 2_AnalyzedData folder contains:

`trained_model/`: the canonical POD-NN surrogate referenced in the paper. Trained on the first 100 simulations (sim_0001..sim_0100), POD fit retains 4 modes, NN trained for up to 500 epochs with random seed 42, batch size 8, 80/20 train/validation split, and the last 400 simulations (sim_1601..sim_2000) held out for testing. Final test metrics: $R^2 = 0.998$, RMSE = 1.94 K, MAE = 1.46 K.

- `pod_nn.h5` — Keras / TensorFlow neural-network weights. Architecture: Input(6) → Dense(64, ReLU, L2=1e-3) → Dropout(0.1) → Dense(64, ReLU, L2=1e-3) → Dropout(0.1) → Dense(32, ReLU, L2=1e-3) → Dense(4). Optimizer Adam (lr=1e-3), loss MSE.
- `pca.pkl` — pickled sklearn PCA object: holds the 4 retained POD spatial modes and the snapshot mean.
- `param_scaler.pkl` — pickled sklearn StandardScaler for the 6 input parameters (mean / std fit on the training set).
- `mode_scaler.pkl` — pickled sklearn StandardScaler for the 4 POD coefficients (mean / std fit on the training-set POD scores).
- `training_history.json` — per-epoch train and validation MSE produced by Keras `model.fit`.

`dataset_size_sensitivity.csv` (script 09) — raw MSE / Max AE / MSE_POD / MSE_NN numbers behind the sensitivity and decomposition plots.

The 3_SourceCode folder contains:

- `_utils.py` — shared module: data loading, POD-NN build / train / predict, save and load of trained artifacts, thermal-resistance computation, common matplotlib styling. All hyperparameters live here as module constants (TRAIN_SIZE=100, TEST_SIZE=400, N_MODES=4, EPOCHS=500, RANDOM_SEED=42, CHIP_AREA=0.0016 m²).
- `01_train_model.py` — trains the canonical POD-NN and writes the artifacts in `2_AnalyzedData/trained_model/`. Run this first.
- `02_pod_modes.py` — POD spatial mode plots.
- `03_pod_reconstruction.py` — original / 1-PC / 4-PC reconstruction quad plot.
- `04_pod_variance.py` — cumulative variance vs mode count.
- `05_mse_history.py` — train / val MSE curve from the saved training history.
- `06_time_crossover.py` — analytical CFD-vs-POD-NN time crossover and per-component bar chart (no data load).
- `07_single_prediction.py` — per-sample prediction / truth / error comparison plots.
- `08_mean_error_field.py` — MAE field on the yz mid-plane across the test set.
- `09_dataset_size_sensitivity.py` — sensitivity sweep (10 trainings) + error decomposition; slow.
- `10_pod_alpha_sweep.py` — POD mode α -sweep heatmaps and per-mode delta plot.
- `11_pearson_correlation.py` — POD coefficient vs physical-quantity Pearson R heatmap.
- `12_rth_latent_space.py` — Rth response surfaces in PC3-PC4.
- `requirements.txt` — minimum dependency list.

Code/software:

The `.npz` files open with `numpy.load(path, allow_pickle=True)` (NumPy ≥ 1.20). The `.pkl` files open with Python's standard `pickle.load` and require scikit-learn to be installed in the same environment (they unpickle into sklearn PCA / StandardScaler instances). The `.h5` file is a Keras model loadable with `tensorflow.keras.models.load_model(path, compile=False)`.

To regenerate every figure from this dataset:

1. Create a Python ≥ 3.10 virtual environment and install the dependencies:

```
pip install -r 3_SourceCode/requirements.txt
```

2. From inside `3_SourceCode/`, run:

```
python 01_train_model.py
```

This must run first; it creates the trained model artifacts in `2_AnalyzedData/trained_model/`. Then run scripts 02 through 12 in any order. Script 09 trains 10 models and takes a few minutes; the others complete in seconds.

Reproducibility note:

Every script seeds NumPy and TensorFlow with `RANDOM_SEED = 42` before training. The first 100 simulations are used for training, the last 400 for testing, and 80/20 train/validation splitting is fixed. With the same Python / TensorFlow versions, repeated runs produce numerically identical results to those in `2_AnalyzedData/`.

Related software (active development):

The current production version of this surrogate-modeling workflow is published at <https://github.com/UARK-NED3/CFDTwin>. CFDTwin is a wizard-driven desktop GUI that automates the same POD-NN pipeline end-to-end against a live ANSYS Fluent case (5-step flow: Setup → DOE → Simulate → Train → Validate).

Important: the data format published here (sim_NNNN.npz + model_setup.json) is a snapshot from an earlier iteration of the codebase and is not directly loadable by the current CFDTwin release. The scripts in 3_SourceCode/ are self-contained and reproduce the paper's results from this static dataset. For new CFD cases, use CFDTwin against your own Fluent setup rather than retrofitting this dataset format.

References:

Curl, D. and Hu, H. "Physically Interpretable Surrogate Modeling of Thermal Fields in Electronics Cooling Using Combined Proper Orthogonal Decomposition and Neural Networks." Department of Mechanical Engineering, University of Arkansas. 2026.