



Poraquê

User Guide

Predicting charge and kinetic energy densities
from crystal geometry

Leandro Seixas

Version 26.6.4

Overview

Poraquê predicts the electronic structure of a crystal from its geometry alone. Given a **POSCAR**, an **INCAR** and a **POTCAR**, it produces the valence charge density and the kinetic energy density — with no wavefunctions and no self-consistency cycle.

It does this by chaining two learned operators:

$$\{\text{POSCAR}, \text{INCAR}, \text{POTCAR}\} \xrightarrow{\text{analytic}} \text{EXTCAR} \xrightarrow{\text{Model 1}} \text{CHGCAR} \xrightarrow{\text{Model 2}} \text{TAUCAR}$$

The first step is closed-form; only the two field-to-field maps are learned. Every output is written in **CHGCAR** format and opens directly in VESTA.

Who this guide is for

This is the practical guide: how to lay out data, train, predict, and read the results. It assumes you can run a plane-wave DFT calculation and want to use Poraquê as a tool.

For the physics and the architecture — why a Fourier neural operator, what the models correspond to in density-functional theory, how the external potential is reconstructed from the pseudopotential tables — see the companion *Technical Guide* in `latex/technical_guide/`.

What you need

Requirement	Note
Python 3.11 or newer	see Chapter 1
A set of DFT calculations	CHGCAR and TAUCAR per structure
An accelerator (optional)	CUDA or Apple Metal; CPU works
<code>latexmk</code> (optional)	only for the automatic PDF reports

Caveat

Poraquê learns from the data you give it. The published results were obtained on five gold supercells, and they measure interpolation between nearby geometries of one element. Nothing in this guide should be read as evidence that a model trained on one chemistry transfers to another — validate on your own held-out structures before trusting a prediction.

Contents

Overview	1
1 Installation	3
1.1 Python version	3
1.2 Checking the installation	3
1.3 Optional components	3
2 Preparing data	4
2.1 Directory layout	4
2.2 Grids may differ between materials	4
2.3 Checking your data	4
3 Training	5
3.1 The short version	5
3.2 One model, all structures	5
3.3 Measuring generalisation	5
3.4 Reading the output	6
3.5 Reference numbers	6
3.6 Options worth knowing	6
4 Predicting a new structure	8
4.1 Choosing the grid	8
4.2 Comparing against a reference	8
4.3 Sanity checks the script performs	9
4.4 Visualising	9
5 Configuration reference	10
5.1 data	10
5.2 model	11
5.3 training	11
5.4 output	11
6 Troubleshooting	12
6.1 The run stops immediately	12
6.2 The results look wrong	12
6.3 Grids and shapes	12
6.4 Devices	13
6.5 Reports	13

CHAPTER 1

Installation

```
git clone https://github.com/seixas-research/poraque.git
cd poraque
pip install -e .
```

This installs NumPy, SciPy, ASE, Matplotlib, PyYAML and PyTorch.

1.1 Python version

Note

Poraquê requires **Python 3.11 or newer**. Every module is verified against the 3.11 grammar. There is deliberately no upper bound, so newer interpreters are supported.

1.2 Checking the installation

```
pytest
python -c "from poraque.ml.device import available_devices;
          ↪ print(available_devices())"
```

The device list is ordered by preference, so the first entry is what `device: auto` will select — `cuda`, then `mps`, then `cpu`.

1.3 Optional components

Component	Needed for	Install
CUDA build of PyTorch	NVIDIA GPUs	https://pytorch.org
pdflatex / latexmk	automatic PDF reports	TeX Live or MacTeX

Apple Silicon needs nothing extra: the Metal backend ships with the standard PyTorch wheel and is selected automatically.

Preparing data

2.1 Directory layout

One directory per material:

```
data/vasp/
  struct_000/  POSCAR  INCAR  POTCAR  CHGCAR  TAUCAR
  struct_001/  POSCAR  INCAR  POTCAR  CHGCAR  TAUCAR
  ...
```

The prefix is configurable (`data.pattern`); anything matching it and containing the required files is picked up.

Implementation

You do not need a modified VASP. The external potential is computed from POSCAR, INCAR and POTCAR using the tabulated pseudopotential, reproducing a reference EXTCAR to a relative 5×10^{-5} . Only CHGCAR and TAUCAR are needed as targets.

2.2 Grids may differ between materials

They usually do: ENCUT and the cell shape fix NGX_F, NGY_F, NGZ_F, so two structures of the same compound can land on different meshes. This is handled — batches are grouped by grid shape and one model serves them all.

Fields are reduced to a common working resolution (`data.resolution`, the longest axis) by Fourier truncation, which is the exact band-limited projection for a plane-wave field: periodicity and the electron count survive to machine precision.

2.3 Checking your data

```
python scripts/validate_vasp_data.py --fit-sigma --form-factor
```

This reads every structure, recomputes the external potential, compares it against a reference EXTCAR if one is present, and runs integrity checks on CHGCAR and TAUCAR — grid agreement, finiteness, sign, and the integrated totals. The electron count is the sharpest check: it should return the exact integer $\sum_a Z_a^{\text{val}}$.

CHAPTER 3

Training

3.1 The short version

```
python scripts/train_fno.py --write-config configs/train_config.yaml
python scripts/train_fno.py --config configs/train_config.yaml
```

This trains **one** ext2chg model and **one** chg2tau model on the combined data of every structure, and writes:

Path	Contents
models/ext2chg.pt, models/chg2tau.pt	the trained operators
logs/*.log, logs/*.json	metrics and full history
logs/*_config.yaml	the resolved configuration
results/plots/	loss curves, cross-sections, parity plots
reports/*.pdf	a typeset report per model

3.2 One model, all structures

Training is *universal*: a single set of weights is fitted to every structure at once, and batches mix materials. That is the deployable artefact.

Caveat

By default nothing is held out, so the metrics printed at the end are **training fit**. They show the model can represent the data; they are not a measure of how it will do on a new material. On the reference data the training fit is 4–5× better than the cross-validated score, which is the size of the mistake if the two are confused.

3.3 Measuring generalisation

Two options, both splitting at the *structure* level.

3.3.1 Hold out named structures

```
python scripts/train_fno.py --config configs/train_config.yaml \
    --holdout struct_004
```

3.3.2 K-fold cross-validation

```
python scripts/train_fno.py --config configs/train_config.yaml \
    --kfold --k-folds 5
```

Each fold trains a fresh model on $K - 1$ groups and scores it on the held-out group. A consolidated PDF reports the mean and standard deviation of each metric across folds.

Implementation

`k_folds` is capped at the number of structures; setting it equal to that count is leave-one-out. Whole materials are held out — never a subset of voxels from a material that also appears in training, which would score the model on data it had effectively already seen.

3.4 Reading the output

Metric	Read it as
relative L^2	fractional error; 0.03 means 3 %
R^2	1 is perfect; <i>negative</i> means worse than the mean
MAE, RMSE	absolute error, in the field's own units
integral error	how well the electron count or T_s is reproduced

For `chg2tau` the log also prints the classical orbital-free functionals (Thomas–Fermi, von Weizsäcker) evaluated on the same input. Beating those is the bar — a learned functional that does not is not adding anything.

3.5 Reference numbers

Five gold supercells, 32^3 working resolution, 200 epochs:

Model	5-fold CV	universal (training fit)
<code>ext2chg</code>	0.0295 ± 0.0025	0.0057
<code>chg2tau</code>	0.0525 ± 0.0031	0.0133

Quote the cross-validated column.

3.6 Options worth knowing

-pauli-head For `chg2tau`, predicts $\tau = \tau_{\text{vW}}[\rho] + s \text{softplus}(f)$ so the exact bound $\tau \geq \tau_{\text{vW}}$ holds by construction. Improves accuracy and trains slightly faster; recommended.

-gaussian-blur Smooths the external potential. Measured to make no meaningful difference at 32^3 and to remove information near the ionic cores; leave it off unless working at higher resolution.

-device `auto`, `cuda`, `mps` or `cpu`. An unavailable backend warns and falls back rather than failing.

-resolution Working grid. Higher is more faithful and much slower; 32 is a reasonable starting point.

CHAPTER 4

Predicting a new structure

```
python scripts/infer_fno.py new_structure/ \  
    --ext2chg models/ext2chg.pt \  
    --chg2tau models/chg2tau.pt \  
    --output predictions/new_structure
```

The directory needs only POSCAR, INCAR and POTCAR. The script computes the external potential, predicts the density, then the kinetic energy density, and writes all three in CHGCAR format.

4.1 Choosing the grid

Resolved in this order, first hit wins:

1. `-grid NX NY NZ` — explicit;
2. `-like FILE` — adopt an existing volumetric file's grid, which is what you want when comparing against a reference calculation;
3. `-resolution N` — longest axis, preserving the aspect ratio;
4. otherwise derived from ENCUT and PREC.

Caveat

A Fourier neural operator is resolution-*flexible*, not resolution-*indifferent*. Evaluating far from the grid density a model was trained on is extrapolation, and no metric printed here can detect it. The script warns when the requested grid departs markedly from the training resolution.

4.2 Comparing against a reference

```
python scripts/infer_fno.py run/ --ext2chg m1.pt --chg2tau m2.pt \  
    --like run/CHGCAR --compare --plot-dir results/plots/check
```

`-compare` scores the prediction against any reference files present, bringing them onto the prediction grid by spectral truncation. `-plot-dir` adds cross-section and parity figures.

4.3 Sanity checks the script performs

- the predicted electron count against $\sum_a Z_a^{\text{val}}$;
- the number of negative density voxels, if any;
- the bound $\tau \geq \tau_{\text{vW}}[\rho]$ on the chained output, with the minimum margin.

A large electron-count error or a violated bound means the prediction should not be trusted, whatever the field looks like.

4.4 Visualising

All outputs are CHGCAR-format and open directly in VESTA. Use `-write-chg` for an additional coarse-formatted CHG.

Configuration reference

A run is defined by a YAML file with four sections. Command-line flags override individual entries, so one committed configuration can be swept from the shell without being edited:

```
python scripts/train_fno.py --config configs/train_config.yaml --epochs 500
```

Note

Unknown keys are *rejected*, not ignored. A typo such as `learn_rate` would otherwise be silently dropped and the run would quietly use the default — producing results that do not match the file that appears to describe them.

The *resolved* configuration is written beside the results, recording what actually ran, overrides included.

5.1 data

Key	Default	Meaning
<code>root</code>	<code>data/vasp</code>	directory of per-material calculations
<code>pattern</code>	<code>struct</code>	prefix identifying those directories
<code>cache</code>	<code>data/cache</code>	where downsampled copies are written
<code>code</code>	<code>auto</code>	DFT code, or detect it
<code>resolution</code>	<code>32</code>	longest grid axis after downsampling
<code>use_vasp_extcar</code>	<code>false</code>	read a reference <code>EXTCAR</code> instead of computing it
<code>gaussian_blur</code>	<code>null</code>	blur width in Å for the computed potential
<code>blur_method</code>	<code>spectral</code>	<code>spectral</code> or <code>ndimage</code>

Caveat

`blur_method: ndimage` blurs along *lattice* axes, which is anisotropic in Cartesian space unless the cell is orthogonal — on a face-centred cubic cell the two methods differ by 2–6 %. Prefer the default.

5.2 model

Key	Default	Meaning
width	16	channel width of the Fourier layers
modes	8	retained modes per axis (a capacity limit)
n_layers	4	number of Fourier layers
projection_channels	48	hidden width of the output head
mode_selection	fixed	fixed index, or physical at constant $G_{\max}$
pauli_residual	false	structural $\tau \geq \tau_{\text{vW}}$ for chg2tau

5.3 training

Key	Default	Meaning
mode	universal	one model on all structures, or leave_one_out
enable_kfold	false	run K-fold cross-validation
k_folds	5	number of folds (capped at the structure count)
holdout	null	structures excluded from universal training
epochs	200	passes over the training set
batch_size	4	capped per grid-shape bucket
learning_rate	0.002	AdamW
device	auto	auto, cuda, mps, cpu
loss	relative_l2	or sobolev (adds a gradient term)
physics	all zero	weights of the physics-informed terms

Physics weights default to zero, so the objective is the plain supervised baseline until one is enabled deliberately. Introduce them one at a time against a measured baseline: a badly scaled constraint degrades accuracy while looking principled.

5.4 output

Key	Default
log, json	logs/...
checkpoint_dir	models
plot_dir	results/plots
report_dir	reports
dpi	160

Troubleshooting

6.1 The run stops immediately

“No struct* directories” Check `data.root` and `data.pattern`. The pattern is a prefix, not a glob.

“use_vasp_extcar is set but ...does not exist” Standard VASP does not write EXTCAR. Unset the flag and let Poraquê compute it.

“No valence charge for ...” No POTCAR was found. Supply one, or pass `-zval Au=11`.

“Unknown key(s) in section ...” A typo in the YAML. The message lists the valid keys.

6.2 The results look wrong

R^2 is negative The model is worse than predicting the mean. Usually too few epochs, or a learning rate that diverged — check the loss curve in `results/plots/`.

The electron count is far off Nothing constrains it by default. A few per cent is normal; tens of per cent means the model is extrapolating, typically because the evaluation grid is far from the training resolution.

The numbers seem too good Check whether anything was held out. In universal mode with `holdout: null` they are training fit. The report’s *What these numbers do not show* section states this explicitly.

6.3 Grids and shapes

“grid shape ...does not match the supplied shared grid” Two fields of one material are on different meshes. All three must share a grid; read them with `grid=` from the same source.

“Cannot batch mixed grid shapes” Use the shape-bucketed loader (the training script does) or `batch_size: 1`.

A few negative voxels in TAUCAR Band-limiting a strictly positive field with sharp core peaks rings slightly. It is an artefact of the downsampling, not of the data, and is why the pipeline uses a sign-tolerant asinh normalisation.

6.4 Devices

“CUDA was requested but ...” A warning, not an error — the run continues on CPU.

Training is slower on MPS than CPU At small grids kernel-launch overhead dominates. The advantage grows with size: $2.2\times$ at 32^3 , $5.7\times$ at 64^3 .

Results differ slightly between devices Expect $\sim 10^{-6}$ relative on a single forward pass. Two *independently trained* runs will differ much more, because optimisation amplifies tiny differences — that is not a bug.

6.5 Reports

A .tex appears instead of a PDF No latexmk or pdflatex on the path. The source is written so it can be compiled elsewhere.

Stray .aux or .log files Should not happen: reports are built in a temporary directory that is deleted wholesale. If you see them, they came from compiling the guides by hand — use `make` in `latex/technical_guide/` or `latex/user_guide/`, which cleans up.