



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 How a value is resolved	10
5.2 Top level	10
5.3 <code>data</code> — where the fields come from	11
5.4 <code>model</code> — the operator's shape	12
5.5 <code>training</code> — how it is fitted	14
5.6 <code>output</code> — what is written	16
5.7 Worked examples	16
6 Troubleshooting	17
6.1 The run stops immediately	17
6.2 The results look wrong	17
6.3 Grids and shapes	17
6.4 Devices	18
6.5 Reports	18

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 `configs/train_config.yaml`: one top-level key and four sections. Generate a copy with every default written out explicitly with

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

This chapter documents every key that file can contain.

5.1 How a value is resolved

Three sources, highest priority first:

1. an explicit command-line flag,
2. the value in the YAML file,
3. the built-in default.

That ordering is what lets a single committed configuration be swept from the shell without being edited or copied:

```
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 dropped silently and the run would quietly use the default — producing results that do not match the file that appears to describe them. The error message names the valid keys for that section.

The *resolved* configuration — defaults, file and command-line overrides merged — is written beside the results as `<json-stem>_config.yaml` (by default `logs/fno_training_config.yaml`). That is the file worth keeping: it records what actually ran, overrides included.

5.2 Top level

Key	Default	Meaning
<code>task</code>	<code>all</code>	Which map to train: <code>ext2chg</code> , <code>chg2tau</code> , or <code>all</code> for both in sequence.

5.3 data — where the fields come from

Key	Default	Meaning
root	data/vasp	Directory holding one subdirectory per material.
cache pattern	data/cache struct	Where downsampled copies are written. Prefix identifying those subdirectories — a prefix, not a glob.
code	auto	DFT code, or <code>auto</code> to detect it from the files present.
resolution	32	Longest grid axis after spectral downsampling.
use_vasp_extcar	false	Read a reference <code>EXTCAR</code> instead of computing one.
gaussian_blur	null	Gaussian blur width in Å applied to the computed potential.
blur_method	spectral	<code>spectral</code> or <code>ndimage</code> .

5.3.1 resolution

Fields are reduced to this many points along their longest axis before training. The reduction is a Fourier truncation — the exact band-limited projection for a plane-wave field — so periodicity is preserved and the electron count is unchanged to machine precision. Interpolation would alias and shift it.

Cost scales as the cube: raising $32 \rightarrow 64$ is roughly eight times the memory and time. Grid *shapes* are reduced in proportion, so a $120 \times 128 \times 128$ calculation becomes $30 \times 32 \times 32$ rather than being forced square.

5.3.2 use_vasp_extcar

Note

Off by default because standard VASP does not write `EXTCAR`. Poraquê reconstructs the external potential from the `POTCAR` tables to a relative error of order 10^{-5} , so the default costs essentially nothing in fidelity and the pipeline runs on any standard VASP output. Setting this to `true` raises an error when the file is absent rather than falling back silently.

5.3.3 gaussian_blur and blur_method

Smooths the computed external potential. The blur is applied *on top of* the tabulated pseudopotential, never in place of it. Both keys are ignored when `use_vasp_extcar` is set, since a reference file is taken as given.

Caveat

`ndimage` uses `scipy.ndimage.gaussian_filter`, which blurs along *grid* axes and is therefore anisotropic in Cartesian space unless the cell is orthogonal — on a face-centred cubic cell the two methods differ by 2–6%. `spectral` multiplies by $e^{-G^2\sigma^2/2}$ using the true reciprocal metric and stays isotropic on any cell. Prefer the default.

Measured at $\sigma = 0.15$ Å and `resolution`: 32, with one structure held out and everything else

fixed, blurring changed the held-out error by 0.7% and the training time by 0.4% — neither meaningful, since the grid already band-limits the field. It does remove information near the ionic cores, where the density peaks. Leave it off unless working at higher resolution.

Implementation

The cache directory name encodes these choices, for example `res32_poraqueext_blur0.15spec`, so changing them cannot silently reuse a cache built with different settings.

5.4 model — the operator’s shape

Key	Default	Meaning
<code>width</code>	16	Channel width of the Fourier layers.
<code>modes</code>	8	Retained Fourier modes per axis.
<code>n_layers</code>	4	Number of Fourier layers.
<code>projection_channels</code>	48	Hidden width of the output projection.
<code>activation</code>	<code>gelu</code>	<code>gelu</code> , <code>relu</code> , <code>silu</code> or <code>tanh</code> .
<code>use_coordinates</code>	<code>true</code>	Append three fractional-coordinate input channels.
<code>cell_conditioning</code>	<code>true</code>	Condition every layer on lattice descriptors (FiLM).
<code>embedding_dim</code>	32	Width of that cell embedding.
<code>mode_selection</code>	<code>fixed</code>	<code>fixed</code> mode index, or <code>physical</code> at constant $G_{\max}$.
<code>g_max</code>	<code>null</code>	Cutoff wavevector in $\text{\AA}^{-1}$; required by <code>mode_selection: physical</code> .
<code>pauli_residual</code>	<code>false</code>	Structural $\tau \geq \tau_{\text{vW}}$ for <code>chg2tau</code> .
<code>pauli_scale</code>	<code>null</code>	Initial Pauli scale in $\text{eV}/\text{\AA}^3$; <code>null</code> fits it from the training split.
<code>learn_pauli_scale</code>	<code>true</code>	Optimise that scale alongside the backbone.

5.4.1 width, modes, n_layers

These three set the capacity. The parameter count is dominated by the spectral weights — $4w^2m^3$ complex numbers per layer for width w and m modes — so `modes` is by far the expensive knob: doubling it multiplies the spectral parameters by eight.

Implementation

`modes` is a *capacity limit*, not a requirement. On a grid too coarse to supply that many modes, fewer are used automatically, so one configuration serves materials whose grids differ in size.

5.4.2 use_coordinates and cell_conditioning

`use_coordinates` appends the three fractional coordinates as extra input channels, giving the network a positional reference the field alone does not carry.

`cell_conditioning` feeds lattice descriptors through an embedding of width `embedding_dim`

and uses it to modulate each layer’s activations (FiLM). Without it the operator sees the field but not the cell it lives in, and cannot distinguish two materials whose grids agree while their lattice vectors do not. Keep both on unless deliberately ablating them.

5.4.3 `mode_selection` and `g_max`

`fixed` keeps the lowest `modes` indices on every material. Because the spacing of reciprocal-lattice points depends on the cell size, that means a *different physical band* for each material.

`physical` instead keeps every mode below a fixed wavevector $G_{\max}$, retaining $\lfloor G_{\max} L_i / 2\pi \rfloor$ modes along axis i , so every material contributes the same band of physics. Prefer it when cell sizes vary widely. It requires `g_max`.

5.4.4 `pauli_residual` and its scale

For `chg2tau`, the model predicts

$$\tau = \tau_{\text{vW}}[\rho] + s \text{softplus}(f_{\theta}(\rho)),$$

so the Hoffmann–Ostenhof bound $\tau \geq \tau_{\text{vW}}$ holds by construction and the network learns only the Pauli term τ_{P} . Against a matched baseline it improved every fold — 18.3% in mean relative L^2 — while training 17.8% *faster*, because the network no longer has to re-derive $|\nabla\rho|^2/8\rho$, roughly 31% of τ . Recommended.

`pauli_scale` sets the initial magnitude s ; `null` fits it from the training split, which is the sensible default. `learn_pauli_scale` lets the optimiser refine it.

Caveat

The bound is a theorem for all-electron densities, whereas `CHGCAR` and `TAUCAR` hold pseudo quantities. Check it on new data before enabling the head — the training log reports any reference points that violate it.

5.5 training — how it is fitted

Key	Default	Meaning
<code>mode</code>	<code>universal</code>	<code>universal</code> or <code>leave_one_out</code> .
<code>holdout</code>	<code>null</code>	Structure names excluded from universal training.
<code>enable_kfold</code>	<code>false</code>	Run K-fold cross-validation; takes precedence over <code>mode</code> .
<code>k_folds</code>	5	Number of folds, capped at the structure count.
<code>epochs</code>	200	Passes over the training set.
<code>batch_size</code>	4	Maximum samples per grid-shape bucket.
<code>learning_rate</code>	0.002	AdamW step size.
<code>weight_decay</code>	0.0001	AdamW weight decay.
<code>scheduler</code>	<code>cosine</code>	<code>cosine</code> decay, or <code>null</code> for a constant rate.
<code>grad_clip</code>	1.0	Global gradient-norm clip; 0 disables.
<code>seed</code>	0	Weight initialisation, batch order and fold shuffling.
<code>device</code>	<code>auto</code>	<code>auto</code> , <code>cuda</code> , <code>mps</code> or <code>cpu</code> .
<code>loss</code>	<code>relative_l2</code>	<code>relative_l2</code> or <code>sobolev</code> .
<code>sobolev_weight</code>	0.1	Gradient-term weight when <code>loss: sobolev</code> .
<code>physics</code>	all 0.0	Physics-informed loss weights; see below.

5.5.1 `mode`, `holdout`, `enable_kfold`, `k_folds`

These four decide *what question the run answers*, and are the settings most worth getting right.

universal Trains **one** model per task on the combined data of every structure and saves a single checkpoint. This is the deployable artefact. Poraquê never trains a separate model per material.

holdout A list of structure names excluded from that training and scored separately, which turns the reported number into a generalisation estimate.

enable_kfold Each fold trains a fresh model on $K - 1$ groups and scores it on the group held out. Produces a generalisation estimate with a spread — but no single model to ship.

leave_one_out The same protocol with K equal to the number of structures.

Both are worth running: cross-validation says whether the architecture generalises, **universal** produces the model that uses all the data.

Caveat

With `mode: universal` and `holdout: null`, nothing is held out and the reported metrics are **training fit**, carrying no generalisation claim. On the reference dataset they are four to five times better than the cross-validated score. Quote the cross-validated numbers.

Implementation

Splitting is always at the *structure* level: whole materials move together. A voxel-level split would place the same crystal on both sides of the split, and since neighbouring voxels are strongly correlated the score would look excellent while saying nothing about transfer to a new material.

5.5.2 batch_size

Capped *per grid-shape bucket*. Materials whose grids differ in shape cannot be stacked into one tensor, so they are grouped by shape and batches drawn within a group. A bucket holding a single material yields batches of one however large this is set. The training log prints the buckets it found.

5.5.3 loss and sobolev_weight

relative_l2 normalises the error per sample, so materials whose fields differ by orders of magnitude contribute equally and the reported value reads directly as a fraction.

sobolev adds a relative gradient term weighted by **sobolev_weight**. Worth enabling when the derivative matters — τ_{vW} depends on $\nabla\rho$, so gradient noise is amplified in low-density regions.

5.5.4 physics

Weight	Constraint it penalises the violation of
electron_count_weight	$\int \rho \, d\mathbf{r} = N$ (ext2chg)
positivity_weight	Non-negativity of the predicted field
von_weizsacker_weight	$\tau \geq \tau_{vW}$ (chg2tau)
euler_lagrange_weight	Orbital-free stationarity residual (ext2chg)

All four default to zero, so the objective is the plain supervised baseline until one is enabled deliberately. Each term is dimensionless, so a single weight can serve a heterogeneous dataset.

Caveat

Introduce them one at a time against a measured baseline, and only once the data term has converged — a physics residual evaluated at random initialisation is noise. A badly scaled constraint degrades accuracy while looking principled.

Where a constraint can be imposed structurally, prefer that: **pauli_residual** enforces $\tau \geq \tau_{vW}$ exactly, whereas **von_weizsacker_weight** only discourages violating it.

5.6 output — what is written

Key	Default	Meaning
log	logs/fno_training.log	Human-readable training log.
json	logs/fno_training.json	Metrics and full loss history.
checkpoint_dir	models	Model weights; <code>null</code> disables checkpointing.
plot_dir	results/plots	Figures; <code>null</code> disables plotting.
report_dir	reports	PDF reports; <code>null</code> disables them.
plot_format	png	png, pdf or svg.
dpi	160	Raster resolution for saved figures.

The resolved configuration is written next to `json` with the suffix `_config.yaml`. PDF reports are assembled in a temporary directory that is removed afterwards, so no `.tex`, `.aux` or `.log` files are left behind.

5.7 Worked examples

```
task: all
model:  {pauli_residual: true}
training: {mode: universal, epochs: 200}
```

```
task: all
training: {enable_kfold: true, k_folds: 5}
```

```
data:      {resolution: 20}
model:     {width: 8, modes: 4, n_layers: 2, projection_channels: 16}
training: {epochs: 10}
```

```
data:      {resolution: 64}
model:     {width: 32, modes: 12, n_layers: 4}
training: {epochs: 400, batch_size: 2}
```

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.