Poraquê

Technical Guide

Density-functional theory and Fourier neural operators
for three-dimensional scalar fields

Leandro Seixas

Version 26.9.15

Preface

Poraquê is a research code for learning maps between the three-dimensional scalar fields of density-functional theory. It has one organising idea: for a given material, the local external potential $V_{\text{ext}}(\mathbf{r})$, the valence charge density $\rho(\mathbf{r})$ and the kinetic energy density $\tau(\mathbf{r})$ all live on *one* grid, in *one* file format, and are therefore directly comparable, composable, and usable as aligned inputs and targets for a neural operator.

Two maps are learned:

$$V_{\text{ext}} \longmapsto \rho \qquad \text{and} \qquad \rho \longmapsto \tau.$$

They are not two unrelated regressions that happen to share a file format. They are the two legs of one variational statement — the first is the Hohenberg–Kohn map, guaranteed to exist by theorem; the second is the kinetic energy density functional, the missing ingredient of orbital-free DFT. Composed, they constitute a complete orbital-free calculation. That relationship is the thread running through this document.

What this guide covers

Chapters 1–2 The physics. Kohn–Sham DFT, the Hohenberg–Kohn theorems, and the orbital-free formulation, with attention to which quantities are exact, which are approximations, and which are the open problem.

Chapter 3 The mathematics of Fourier neural operators: why an operator between function spaces is the right object, how the spectral convolution is constructed, and why it is the natural architecture for a periodic crystal on a plane-wave grid.

Chapters 4–5 Implementation. The shared-grid data model, the code-agnostic ingestion layer, and a detailed account of the interface to VASP — including the exact reconstruction of the local pseudopotential.

Chapter 6 Measured results on a platinum data set, with the analytic baselines a learned functional must beat.

Chapter 7 The physics-informed programme: how the governing equations re-enter as constraints, and which of them are already enforced by construction.

Conventions

Two unit systems appear, deliberately.

- **Å and eV** in everything that touches a file. Every plane-wave code writes geometry in Ångström and energies in electronvolt, so the field layer, the readers and the machine-learning code use those units throughout. A charge density is in $e/\text{Å}^3$, a kinetic energy

density in $\mathrm{eV}/\mathrm{Å}^3$, a potential in eV.

- **Hartree atomic units** in the analytical expressions of Chapters 1–2, where $\hbar = m_e = e = 1$ removes constants that would otherwise clutter every equation.

Conversions are collected in `poraque/fields/constants.py`. The two constants that recur are

$$\frac{e^2}{4\pi\varepsilon_0} = 14.399645 \text{ eV} \cdot \text{Å}, \qquad \frac{\hbar^2}{2m_e} = 3.809982 \text{ eV} \cdot \text{Å}^2.$$

Sign convention for potentials: $V_{\text{ext}}(\mathbf{r})$ is the potential *energy of an electron*, so it is negative near the ions. This matches VASP's `LOCPOT` and `EXTCAR`.

> **Note**
> Passages marked **Implementation** connect an equation to the function that evaluates it. Passages marked **Caveat** record an approximation, an untested path, or a limit on what the numbers presented here support. They are not disclaimers added for politeness: each one marks a place where a reader could otherwise draw a conclusion the evidence does not carry.

Contents

Preface **1**

1 Kohn–Sham density-functional theory **5**
1.1 The many-body problem . . . 5
1.2 The Hohenberg–Kohn theorems . . . 5
1.3 The Kohn–Sham construction . . . 6
1.4 The kinetic energy density . . . 6
1.5 Exact properties of τ . . . 7
1.6 Pseudopotentials . . . 7

2 Orbital-free density-functional theory **9**
2.1 The variational problem . . . 9
2.2 The kinetic energy functional problem . . . 9
2.3 Functional derivatives . . . 10
2.4 Electrostatics on a periodic grid . . . 10
2.5 The external potential of pseudo-ions . . . 10

3 Fourier neural operators **12**
3.1 Learning operators, not functions . . . 12
3.2 Kernel integral operators . . . 12
3.3 The Fourier layer . . . 12
3.4 Discretisation invariance . . . 13
3.5 Handling grids that differ between materials . . . 14
3.6 Cell conditioning . . . 14
3.7 Architecture summary . . . 14
3.8 The five parameters that size a run . . . 15

4 Implementation **22**
4.1 The shared-grid data model . . . 22
4.2 Grid construction . . . 22
4.3 Spectral resampling . . . 23
4.4 Code-agnostic ingestion . . . 23
4.5 Hardware acceleration . . . 24
4.6 A C kernel for CPU inference . . . 24
4.7 Numerical precision . . . 26
4.8 Configuration . . . 26

5 The VASP interface **27**
5.1 `EXTCAR`: the local pseudopotential . . . 27
5.2 `TAUCAR`: the kinetic energy density . . . 29
5.3 What is absent from `EXTCAR` . . . 30
5.4 When the pseudopotentials are absent . . . 30

6 Results **32**
6.1 The data set . . . 32
6.2 Verification before learning . . . 32
6.3 Model 1: $V_{\text{ext}} \mapsto \rho$ . . . 33
6.4 Model 2: $\rho \mapsto \tau$ . . . 33
6.5 The inference pipeline . . . 34
6.6 Hardware . . . 34
6.7 What these numbers do and do not support . . . 35

7 Physics-informed operators **36**
7.1 Constrain by construction, not by penalty . . . 36
7.2 Measured effect of the head . . . 37
7.3 Soft constraints . . . 38
7.4 The closed loop . . . 38
7.5 Why this is cheap here . . . 39
7.6 Roadmap . . . 39

8 The role of Model 2: a learned kinetic energy functional **40**
8.1 The question, stated without charity . . . 40
8.2 Is τ a functional of ρ? . . . 40
8.3 What the model must get right — and it is not τ . . . 41
8.4 The mechanism: $\delta T_{\text{s}}/\delta\rho$ by autograd . . . 41
8.5 Coupling: how Model 2 trains Model 1 . . . 42
8.6 Critical assessment . . . 43
8.7 Verification plan . . . 44
8.8 Status . . . 45
8.9 Distilling the Pauli factor, and which candidate to quote . . . 45

9 The training pipeline **47**
9.1 Requirements . . . 47
9.2 Configuration . . . 47
9.3 The external potential is computed, not imported . . . 48
9.4 Gaussian smoothing of the potential . . . 48
9.5 Training . . . 49
9.6 K-fold cross-validation . . . 50
9.7 Automatic PDF reporting . . . 50
9.8 Outputs . . . 51

10 From fields to energies **52**
10.1 The expression . . . 52
10.2 The $\mathbf{G} = 0$ bookkeeping . . . 53
10.3 Ewald . . . 54
10.4 What the number means . . . 54
10.5 The ASE interface . . . 56

CHAPTER 1

Kohn–Sham density-functional theory

This chapter fixes the physical objects the rest of the guide manipulates. The emphasis is on *which statements are theorems and which are approximations*, because that distinction determines what a neural network may legitimately be asked to learn, and what may be enforced on it as a hard constraint.

1.1 The many-body problem

The non-relativistic electronic Hamiltonian for N electrons in the field of fixed nuclei is, in Hartree atomic units,

$$\hat{H} = -\tfrac{1}{2}\sum_{i}^{N} \nabla_i^2 + \sum_{i}^{N} V_{\text{ext}}(\mathbf{r}_i) + \sum_{i<j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}, \tag{1.1}$$

where V_{ext} collects the electron–nucleus attraction. The wavefunction $\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N)$ depends on $3N$ coordinates, so a direct solution is hopeless beyond a handful of electrons.

1.2 The Hohenberg–Kohn theorems

Density-functional theory replaces Ψ by the electron density

$$\rho(\mathbf{r}) = N \int |\Psi(\mathbf{r}, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2 \, \mathrm{d}\mathbf{r}_2 \cdots \mathrm{d}\mathbf{r}_N, \tag{1.2}$$

a function of three coordinates. Two theorems justify this.

First theorem. The ground-state density determines the external potential up to an additive constant. Consequently, for a fixed electron number, the ground-state density is a functional of V_{ext} alone:

$$V_{\text{ext}}(\mathbf{r}) \;\longleftrightarrow\; \rho_0(\mathbf{r}). \tag{1.3}$$

Second theorem. There exists a universal functional $F[\rho]$ such that the total energy

$$E[\rho] = F[\rho] + \int V_{\text{ext}}(\mathbf{r})\rho(\mathbf{r}) \, \mathrm{d}\mathbf{r} \tag{1.4}$$

is minimised, subject to $\int \rho \, \mathrm{d}\mathbf{r} = N$, by the ground-state density, where it equals the ground-state energy.

Equation (1.3) is the formal justification for the first learned model of this work. The target of a $V_{\text{ext}} \mapsto \rho$ regression is not a fitted correlation: it is a single-valued function whose existence and uniqueness are guaranteed. That is a stronger footing than most machine-learning targets enjoy.

Caveat

The map is conditional on the electron number N. In the present data set $N = \sum_a Z_a^{\mathrm{val}}$ follows from the geometry, so a network can infer it; for charged systems or a chemically heterogeneous data set, N must become an explicit input.

1.3 The Kohn–Sham construction

The difficulty is that $F[\rho]$ is unknown, and its dominant part — the kinetic energy — is badly approximated by explicit density functionals. Kohn and Sham sidestep this by introducing an auxiliary system of non-interacting electrons with the *same* density, for which the kinetic energy can be computed exactly from orbitals. Writing

$$F[\rho] = T_{\mathrm{s}}[\rho] + E_{\mathrm{H}}[\rho] + E_{\mathrm{xc}}[\rho], \tag{1.5}$$

with the Hartree energy

$$E_{\mathrm{H}}[\rho] = \frac{1}{2} \iint \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{r}', \tag{1.6}$$

the variational condition becomes a set of single-particle equations,

$$\left(-\tfrac{1}{2}\nabla^2 + v_{\mathrm{s}}[\rho](\mathbf{r})\right) \psi_i(\mathbf{r}) = \varepsilon_i \psi_i(\mathbf{r}), \qquad v_{\mathrm{s}} = V_{\mathrm{ext}} + v_{\mathrm{H}}[\rho] + v_{\mathrm{xc}}[\rho], \tag{1.7}$$

with

$$\rho(\mathbf{r}) = \sum_i^{\mathrm{occ}} f_i |\psi_i(\mathbf{r})|^2, \qquad T_{\mathrm{s}} = -\tfrac{1}{2} \sum_i^{\mathrm{occ}} f_i \langle \psi_i | \nabla^2 | \psi_i \rangle. \tag{1.8}$$

Because v_{s} depends on ρ, which depends on the orbitals, Eq. (1.7) must be solved self-consistently.

1.3.1 Why the map is hard

Two features of Eq. (1.7) shape the choice of neural architecture in Chapter 3.

1. **It is non-local.** The Hartree term is a $1/|\mathbf{r} - \mathbf{r}'|$ convolution: the density at one point depends on the potential everywhere. A network with a finite receptive field must be made very deep to propagate that information across a unit cell.

2. **It is self-consistent.** The density is a fixed point, not a single forward evaluation. A supervised model learns the fixed point directly, which is what makes it fast — and what makes it fragile off the training distribution.

1.4 The kinetic energy density

The quantity stored in `TAUCAR` is the *positive-definite* kinetic energy density of the Kohn–Sham system,

$$\tau(\mathbf{r}) = \frac{1}{2} \sum_i^{\mathrm{occ}} f_i \left|\nabla \psi_i(\mathbf{r})\right|^2. \tag{1.9}$$

It integrates to T_{s} and is manifestly non-negative. An alternative definition using the Laplacian, $\tilde{\tau} = -\frac{1}{2} \sum_i f_i \psi_i^* \nabla^2 \psi_i$, integrates to the same T_{s} but differs pointwise by $\frac{1}{4}\nabla^2 \rho$ and is *not* sign-definite. The distinction matters here: the constraints of Section 1.5 apply to Eq. (1.9), and applying them to the Laplacian form would be simply wrong.

Implementation

Chapter 5 shows, from the VASP source, that `TAUCAR` indeed stores Eq. (1.9) — built spectrally as $i(\mathbf{G}+\mathbf{k})$ acting on the plane-wave coefficients — and not the Laplacian form. This was verified rather than assumed.

1.5 Exact properties of τ

Several relations constrain τ exactly. They are theorems, and Chapter 7 uses them as hard constraints rather than as penalties.

Non-negativity. $\tau(\mathbf{r}) \geq 0$, immediately from Eq. (1.9).

The von Weizsäcker bound. For any system,

$$\tau(\mathbf{r}) \;\geq\; \tau_{\mathrm{vW}}(\mathbf{r}) \;\equiv\; \frac{|\nabla\rho(\mathbf{r})|^2}{8\rho(\mathbf{r})}, \tag{1.10}$$

with equality for a single nodeless orbital. This is the Hoffmann-Ostenhof inequality, and it is the single most useful constraint available for the $\rho \mapsto \tau$ map: τ_{vW} is a closed-form functional of the input, so the bound can be imposed *by construction*.

Uniform-gas limit. As $\nabla\rho \to 0$,

$$\tau \;\longrightarrow\; \tau_{\mathrm{TF}} = C_{\mathrm{TF}}\,\rho^{5/3}, \qquad C_{\mathrm{TF}} = \tfrac{3}{10}(3\pi^2)^{2/3} \approx 2.8712. \tag{1.11}$$

Coordinate scaling. For $\rho_\lambda(\mathbf{r}) = \lambda^3\rho(\lambda\mathbf{r})$,

$$T_{\mathrm{s}}[\rho_\lambda] = \lambda^2\, T_{\mathrm{s}}[\rho]. \tag{1.12}$$

This is exact and label-free: any training sample can be rescaled and its target T_{s} is then known in closed form, which is a data augmentation at zero DFT cost.

The decomposition suggested by Eq. (1.10),

$$\tau = \tau_{\mathrm{vW}}[\rho] + \tau_{\mathrm{P}}, \qquad \tau_{\mathrm{P}} \geq 0, \tag{1.13}$$

defines the *Pauli term* τ_{P}. It is the part of the kinetic energy that arises purely from Fermi statistics — what distinguishes a many-fermion system from a single-orbital one — and it is the object every orbital-free kinetic functional is really trying to approximate. Section 7.1.3 shows that learning τ_{P} instead of τ is both more accurate and cheaper.

1.6 Pseudopotentials

Core electrons are chemically inert but carry sharp oscillations that would demand an impractically fine grid. A pseudopotential replaces the nucleus and core by an effective ion of charge Z^{val} acting on smooth pseudo-valence orbitals. Its local part behaves as $-Z^{\mathrm{val}}e^2/r$ outside a core radius R_{core} and is softened inside.

Three consequences run through this document:

1. The fields in `CHGCAR` and `TAUCAR` are *pseudo* quantities. They are mutually consistent, which is why the bound (1.10) is observed to hold on this data even though it is strictly guaranteed only for all-electron quantities.

2. The external potential in `EXTCAR` is the *local* part only. Non-local projectors are not functions of $\mathbf{r}$ at all and cannot appear in a volumetric file even in principle.

3. The short-range shape of the local pseudopotential is tabulated, not analytic. Chapter 5 shows that modelling it with a smooth analytic form factor is the dominant error in a naive reconstruction, and how to avoid it.

CHAPTER 2

Orbital-free density-functional theory

Kohn–Sham theory buys an exact kinetic energy at the price of reintroducing N orbitals, and with them the $\mathcal{O}(N^3)$ cost of orthogonalising and diagonalising. Orbital-free DFT removes the orbitals again. It is the reason the second learned map matters.

2.1 The variational problem

Return to Eq. (1.4) and write the energy purely as a functional of the density,

$$E[\rho] = T_\mathrm{s}[\rho] + \int V_\mathrm{ext}(\mathbf{r})\,\rho(\mathbf{r})\,\mathrm{d}\mathbf{r} + E_\mathrm{H}[\rho] + E_\mathrm{xc}[\rho]. \tag{2.1}$$

Minimising under the constraint $\int \rho\,\mathrm{d}\mathbf{r} = N$ with a Lagrange multiplier μ gives the Euler–Lagrange equation

$$\boxed{\frac{\delta T_\mathrm{s}}{\delta\rho}(\mathbf{r}) + V_\mathrm{ext}(\mathbf{r}) + v_\mathrm{H}[\rho](\mathbf{r}) + v_\mathrm{xc}[\rho](\mathbf{r}) \;=\; \mu} \tag{2.2}$$

with μ the chemical potential — a *constant* in space. A single scalar equation replaces the N coupled equations (1.7), and the cost becomes essentially linear in system size.

Equation (2.2) is the central equation of this work. Every symbol in it is something Poraquê has:

Term	Source
V_ext	the input of Model 1 (`EXTCAR`)
ρ	the output of Model 1 (`CHGCAR`)
$v_\mathrm{H}[\rho]$	computed *exactly* by FFT, $4\pi e^2\rho(\mathbf{G})/G^2$
$\delta T_\mathrm{s}/\delta\rho$	autograd through Model 2 ($T_\mathrm{s} = \int \tau$)
μ	eliminated by removing the cell average

That is what allows Eq. (2.2) to serve as a *label-free* training signal in Chapter 7.

2.2 The kinetic energy functional problem

Everything in Eq. (2.1) has an accurate explicit density functional except $T_\mathrm{s}[\rho]$. The classical approximations are:

Thomas–Fermi. $T_\mathrm{s}^\mathrm{TF}[\rho] = C_\mathrm{TF}\int \rho^{5/3}$, exact for the uniform electron gas. It has no shell structure and does not bind molecules.

von Weizsäcker. $T_\mathrm{s}^\mathrm{vW}[\rho] = \int |\nabla\rho|^2/8\rho$, exact for one orbital, a rigorous lower bound in general.

Gradient expansions. $T_s^{\mathrm{TF}} + \lambda T_s^{\mathrm{vW}}$ with $\lambda = 1/9$ (second order) or $\lambda = 1$ (strongly inhomogeneous densities). Neither is reliable across chemistry.

Chapter 6 measures these on platinum. Thomas–Fermi achieves a *negative* coefficient of determination — it is a worse pointwise predictor of τ than the constant mean — which is the expected failure of a uniform-gas limit applied to a d-band metal. That is the gap a learned functional is meant to close.

2.3 Functional derivatives

The quantity Eq. (2.2) actually requires is not τ but $\delta T_s/\delta\rho$. For the two classical forms,

$$\frac{\delta T_s^{\mathrm{TF}}}{\delta\rho} = \tfrac{5}{3}\, C_{\mathrm{TF}}\, \rho^{2/3}, \tag{2.3}$$

$$\frac{\delta T_s^{\mathrm{vW}}}{\delta\rho} = -\frac{1}{2}\frac{\nabla^2\sqrt{\rho}}{\sqrt{\rho}}. \tag{2.4}$$

Caveat

A model with small error in τ may still have a poor functional derivative, because differentiation amplifies high-frequency error. Training on τ alone therefore optimises a proxy for the quantity that is used. Chapter 7 describes training *through* the derivative to remove this mismatch.

2.4 Electrostatics on a periodic grid

Both V_{ext} and v_{H} are Coulombic, and both are evaluated in reciprocal space, where the kernel is diagonal. Solving $\nabla^2 v_{\mathrm{H}} = -4\pi e^2\rho$ by Fourier transform gives

$$v_{\mathrm{H}}(\mathbf{G}) = \frac{4\pi e^2\,\rho(\mathbf{G})}{G^2}, \qquad v_{\mathrm{H}}(\mathbf{G}=0) \equiv 0. \tag{2.5}$$

The $\mathbf{G} = 0$ component diverges for any charged distribution. In a periodic crystal this divergence cancels exactly between the electron–electron, electron–ion and ion–ion terms, provided the cell is neutral overall. The standard convention — adopted here and by VASP — is to set the $\mathbf{G} = 0$ component of *each* Coulomb term to zero individually, which amounts to adding a uniform neutralising background. The consequence is that potentials are defined only up to a constant, and their cell average vanishes.

2.5 The external potential of pseudo-ions

For pseudo-ions of charge Z_s^{val} at positions $\boldsymbol{\tau}_a$, the local external potential is assembled in reciprocal space as

$$V_{\mathrm{ext}}(\mathbf{G}) = -\frac{4\pi e^2}{\Omega}\sum_s \frac{Z_s^{\mathrm{val}} f_s(G)}{G^2}\, S_s(\mathbf{G}), \qquad S_s(\mathbf{G}) = \sum_{a\in s} e^{-i\mathbf{G}\cdot\boldsymbol{\tau}_a}, \tag{2.6}$$

with $V_{\text{ext}}(\mathbf{G} = 0) \equiv 0$ and Ω the cell volume. The real-space field follows from one inverse FFT. The construction is $\mathcal{O}(N \log N)$, exactly periodic, and free of any real-space cutoff or Ewald parameter.

The form factor $f_s(G)$ encodes the pseudisation:

$f_s = 1$ bare point ions — the exact long-range limit, but its $1/G^2$ tail is truncated at the Nyquist frequency of any finite grid, producing visible ringing near the nuclei.

$f_s = e^{-G^2\sigma_s^2/2}$ Gaussian ions, equivalent to $-Z_s^{\text{val}} e^2 \operatorname{erf}(r/\sqrt{2}\sigma_s)/r$ in real space: band-limited, smooth, and controlled by one width per species.

f_s **tabulated** the true pseudopotential form factor, read from the `POTCAR`. Chapter 5 shows this is the only choice that reproduces a reference calculation, because the true $f_s(G)$ *oscillates about zero* at large G and no monotonic analytic form can represent it.

> **Implementation**
>
> Implemented in `poraque.fields.ExternalPotential`. The structure factors are built from separable one-dimensional phase vectors — since $\mathbf{G} \cdot \boldsymbol{\tau}_a = 2\pi \sum_j m_j s_{aj}$ for FFT frequencies m_j and fractional coordinates s_{aj} — so the cost is $\mathcal{O}(N_{\text{at}}(N_1+N_2+N_3))$ complex exponentials rather than $\mathcal{O}(N_{\text{at}} N_1 N_2 N_3)$, with no full complex grid allocated per atom.

Verification of Eq. (2.6) is by Poisson's equation: the spectral Laplacian of the constructed field is compared against $4\pi e^2(n_{\text{ion}} - \langle n_{\text{ion}} \rangle)$, with n_{ion} built independently by a real-space Gaussian lattice sum. The residual is 4.5×10^{-12} against a scale of 159 eV/Å^2 — machine precision, and an independent code path.

CHAPTER 3

Fourier neural operators

3.1 Learning operators, not functions

A conventional network learns a map between finite-dimensional vectors. Retrain it for a different input size and nothing transfers. That is fatal here: the grid (N_1, N_2, N_3) of a plane-wave calculation is fixed by the cell and the cutoff, so *every material has a different one.*

A *neural operator* instead learns a map between function spaces,

$$\mathcal{G} : \mathcal{A} \longrightarrow \mathcal{U}, \qquad a(\mathbf{r}) \longmapsto u(\mathbf{r}), \tag{3.1}$$

and is then discretised only for evaluation. Its parameters describe the operator, not a particular sampling of it, so one set of weights serves every grid.

3.2 Kernel integral operators

The construction generalises the affine layer. Lift the input pointwise to a higher-dimensional channel space, $v_0(\mathbf{r}) = P\,a(\mathbf{r})$, apply L layers of the form

$$v_{\ell+1}(\mathbf{r}) = \sigma\big(W v_\ell(\mathbf{r}) + b + (\mathcal{K}(v_\ell))(\mathbf{r})\big)\,, \tag{3.2}$$

and project back pointwise, $u(\mathbf{r}) = Q\,v_L(\mathbf{r})$. Here W is a local (channel-mixing) linear map and $\mathcal{K}$ is a *kernel integral operator*

$$(\mathcal{K}v)(\mathbf{r}) = \int_D \kappa(\mathbf{r}, \mathbf{r}')\, v(\mathbf{r}')\, \mathrm{d}\mathbf{r}'. \tag{3.3}$$

The pointwise terms alone would give a network with no spatial coupling at all; $\mathcal{K}$ is what makes it an operator.

3.3 The Fourier layer

Restricting the kernel to be translation invariant, $\kappa(\mathbf{r}, \mathbf{r}') = \kappa(\mathbf{r} - \mathbf{r}')$, turns the integral into a convolution, which the convolution theorem diagonalises:

$$(\mathcal{K}v)(\mathbf{r}) = \mathcal{F}^{-1}[R(\mathbf{G}) \cdot \mathcal{F}[v](\mathbf{G})]\,(\mathbf{r}), \tag{3.4}$$

where $R(\mathbf{G})$ is a learned complex multiplier. Rather than parameterising κ in real space, the Fourier neural operator parameterises R directly and *truncates* it to the lowest $M_1 \times M_2 \times M_3$ modes.

Truncation does three things at once: it regularises (high frequencies are where noise lives), it bounds the parameter count, and — decisively — it makes the weight tensor independent of the grid. Nothing in R refers to (N_1, N_2, N_3).

3.3.1 Why this suits a crystal

Three properties align the architecture with the physics of Chapters 1–2.

Periodicity is exact, not learned. The FFT imposes Born–von Kármán boundary conditions, which a crystal unit cell already obeys. A padded convolutional network would have to spend capacity imitating that, and would still produce edge artefacts.

The coupling is global in one layer. Multiplying in reciprocal space is convolving over the whole cell in real space. This directly addresses the non-locality of Section 1.2: the Hartree kernel $4\pi e^2/G^2$ that the operator must partly represent is *diagonal* in exactly the basis the layer works in.

Differential operators are free and exact. On a plane-wave grid $\nabla \to i\mathbf{G}$ and $\nabla^2 \to -G^2$ are exact for band-limited fields. Since the layer already computes FFTs, the physics-informed terms of Chapter 7 cost almost nothing — and carry no discretisation error that the loss would otherwise misattribute to the network.

3.4 Discretisation invariance

For a real input the transform is taken with `rfftn`, halving the last axis. The retained coefficients form four "corners" in the first two axes — combinations of low positive and low negative frequencies — each with its own weight block. The parameter count is

$$4 \times C_{\text{in}} \times C_{\text{out}} \times M_1 M_2 M_3 \quad \text{complex numbers,} \tag{3.5}$$

independent of the grid.

Two further choices are required for weights to be *meaningful* across resolutions, not merely applicable.

3.4.1 Normalisation

The discrete transform must approximate the continuous Fourier series. With coefficients

$$c_{\mathbf{G}} = \frac{1}{N} \sum_n u_n e^{-i\mathbf{G}\cdot\mathbf{r}_n}, \tag{3.6}$$

i.e. `norm="forward"` in both directions, $c_{\mathbf{G}}$ converges to the continuous series coefficient as $N \to \infty$ and its magnitude does not drift with N. Under the default convention a 120^3 field would enter a layer with amplitudes $\sim 125\times$ those of a 24^3 field and shared weights would be meaningless.

> **Implementation**
> Measured: a band-limited field sampled at 16^3 and 32^3 gives operator outputs agreeing to a relative 1.5×10^{-6} at the shared points.

3.4.2 Mode selection

Truncating at a fixed mode *index* keeps a different physical wavevector band for every cell size, since $|\mathbf{G}_{\max}| = 2\pi M/L$. Truncating instead at a fixed $G_{\max}$ — retaining

$$M_i = \left\lfloor \frac{G_{\max} L_i}{2\pi} \right\rfloor \tag{3.7}$$

modes — makes the operator act on the same band of physics in every material. Poraquê offers both; the physical selection is preferred when the data set spans a wide range of cell sizes.

3.5 Handling grids that differ between materials

Three mechanisms together deliver true grid independence.

1. **Dynamic mode truncation.** Weights are allocated for M_i modes; each forward pass uses $\min(M_i, \text{modes available on this grid})$ and leaves the rest untouched. A 24^3 and a 120^3 sample share the same parameters.

2. **Resolution-invariant normalisation**, as above.

3. **Shape-bucketed batching.** Samples of different spatial shape cannot be stacked. Rather than pad — which wastes compute and injects fictitious vacuum into the FFT — materials are grouped by (N_1, N_2, N_3) and batches drawn within a group.

Normalisation layers must also be shape-agnostic: `GroupNorm` is used throughout, since batch statistics are meaningless when every sample has a different spatial extent.

3.6 Cell conditioning

Two materials can share a grid shape and differ entirely in physical scale, so the lattice must enter explicitly. Poraquê supplies it twice: as three fractional-coordinate channels appended to the input (bounded in $[0, 1)$ for every material), and through FiLM conditioning, in which a small network maps rotation-invariant cell descriptors — the three lattice lengths, the three angle cosines, and $\Omega^{1/3}$ — to per-channel scale and shift parameters. Neither touches the spatial dimensions, so both are oblivious to the grid.

3.7 Architecture summary

Stage	Operation
Input	$(B, 1, N_1, N_2, N_3)$, normalised field
Coordinates	append 3 fractional-coordinate channels
Lifting	$1 \times 1 \times 1$ convolution to width C
Fourier layers ($\times L$)	Eq. (3.4) + pointwise + GroupNorm + FiLM, residual
Projection	two $1 \times 1 \times 1$ convolutions
Output	$(B, 1, N_1, N_2, N_3)$

The residual connection around each Fourier layer stabilises deeper stacks; the pointwise branch supplies the local, high-frequency content that mode truncation removes from the spectral branch.

3.8 The five parameters that size a run

Four settings fix the operator's shape and one fixes the grid it is trained on. They are easy to confuse, because four of the five are "some number of channels" and the fifth is the only one that is not a property of the model at all. Written against the architecture above:

Key	Symbol	Section	What it sets
`model.width`	C	`model`	channels carried through the stack
`model.modes`	M_i	`model`	Fourier coefficients kept per axis
`model.n_layers`	L	`model`	how many times the cycle runs
`model.projection_channels`	C_P	`model`	hidden width of the read-out
`data.resolution`	N	`data`	longest grid axis after downsampling

3.8.1 The operator, with every index written out

Equations (3.2) and (3.4) suppress the channel indices, which is what makes the four architecture settings hard to tell apart: they are all "some number of channels" until it is written down which sum each one is the upper limit of. Written out in full, with $\mathbf{r}$ a grid point and $\mathbf{k}$ a reciprocal-lattice point:

Lifting.

$$v_o^{(0)}(\mathbf{r}) = \sum_{i=1}^{C_{\text{in}}+3} W_{oi}^{\text{lift}}\, x_i(\mathbf{r}) + b_o^{\text{lift}}, \qquad o = 1, \dots, C \tag{3.8}$$

The $+3$ is the appended fractional coordinates. No nonlinearity here: the lift is affine.

Fourier layer $\ell = 1, \dots, L$.

$$\hat{v}_i^{(\ell-1)}(\mathbf{k}) = \mathcal{F}\big[v_i^{(\ell-1)}\big](\mathbf{k}) \tag{3.9}$$

$$\hat{z}_o^{(\ell)}(\mathbf{k}) = \begin{cases} \displaystyle\sum_{i=1}^{C} R_{io\,k_1k_2k_3}^{(\ell,\,c(\mathbf{k}))}\, \hat{v}_i^{(\ell-1)}(\mathbf{k}) & \mathbf{k} \text{ retained}, \\ 0 & \text{otherwise}, \end{cases} \tag{3.10}$$

$$z_o^{(\ell)}(\mathbf{r}) = \mathcal{F}^{-1}\big[\hat{z}_o^{(\ell)}\big](\mathbf{r}) \tag{3.11}$$

$$y_o^{(\ell)}(\mathbf{r}) = \text{GroupNorm}\Big(z_o^{(\ell)}(\mathbf{r}) + \sum_{i=1}^{C} W_{oi}^{(\ell)}\, v_i^{(\ell-1)}(\mathbf{r}) + b_o^{(\ell)}\Big) \tag{3.12}$$

$$y_o^{(\ell)}(\mathbf{r}) \leftarrow \big(1 + \gamma_o(\text{cell})\big)\, y_o^{(\ell)}(\mathbf{r}) + \beta_o(\text{cell}) \tag{3.13}$$

$$v_o^{(\ell)}(\mathbf{r}) = v_o^{(\ell-1)}(\mathbf{r}) + \sigma\big(y_o^{(\ell)}(\mathbf{r})\big) \tag{3.14}$$

Read-out.

$$u_p(\mathbf{r}) = \sigma\Big(\sum_{o=1}^{C} P_{po}^{(1)}\, v_o^{(L)}(\mathbf{r}) + b_p^{(1)}\Big), \qquad p = 1, \dots, C_P \tag{3.15}$$

$$\text{out}_q(\mathbf{r}) = \sum_{p=1}^{C_P} P_{qp}^{(2)}\, u_p(\mathbf{r}) + b_q^{(2)}, \qquad q = 1, \dots, C_{\text{out}} \tag{3.16}$$

with the index ranges

Index	Range	Set by
o, i	$1, \dots, C$	`width`
ℓ	$1, \dots, L$	`n_layers`
p	$1, \dots, C_P$	`projection_channels`
k_1, k_2, k_3	$k_1 < m_1,\ k_2 < m_2,\ 0 \le k_3 < m_3$	`modes`, with `resolution`
c	$1, \dots, 4$	the four $(\pm k_1, \pm k_2)$ corners
q	$1, \dots, C_{\text{out}}$	the task

and the retained band, for a sample on an (N_1, N_2, N_3) grid,

$$m_1 = \min\Big(M, \tfrac{N_1}{2}\Big), \quad m_2 = \min\Big(M, \tfrac{N_2}{2}\Big), \quad m_3 = \min\Big(M, \tfrac{N_3}{2} + 1\Big). \tag{3.17}$$

The third axis is the one `rfftn` halves, so it carries only $k_3 \ge 0$; the first two carry both signs, which is what the four corners c enumerate.

3.8.2 GroupNorm, FiLM, and where the nonlinearity goes

Equations (3.12)–(3.14) are three steps between the inverse transform and the next layer's input, and their *order* is load-bearing. The canonical Fourier neural operator has none of them: Li *et al.* write $v_{\ell+1} = \sigma(W v_\ell + \mathcal{K} v_\ell)$, the activation applied directly to the sum. Poraquê inserts a normalisation and a conditioning step in between, and closes with a residual.

GroupNorm — because the batch is not a population. The spectral branch's output scale is not controlled by anything: it is a sum over $m_1 m_2 m_3$ retained coefficients of a field whose spectrum varies from material to material, so two samples in the same batch can leave Eq. (3.11) an order of magnitude apart. Something has to set the scale the activation sees.

It cannot be `BatchNorm`. Batch statistics assume the samples in a batch are draws from one population, and here they are not even the same *shape*: shape-bucketed batching (Section 3.4) means the batch is whatever materials happen to share a grid, its size varies, and a 24^3 and a 120^3 sample never meet. `GroupNorm` computes its statistics *within a sample*, over a group of channels and all of space, so it is indifferent to both.

The group count is the largest divisor of C not exceeding 8 — at $C = 16$, eight groups of two channels — because `GroupNorm` requires divisibility and `width` is free.

FiLM — the only place the lattice reaches the stack. Two materials can share a grid shape and differ entirely in physical scale, so the cell must enter explicitly. Feature-wise linear modulation is one affine map per channel, driven by the cell embedding:

$$y_o^{(\ell)} \longleftarrow (1 + \gamma_o(\text{cell}))\, y_o^{(\ell)} + \beta_o(\text{cell}), \tag{3.18}$$

which is Eq. (3.13). It touches no spatial index, so it is oblivious to the grid — and the projection producing γ and β is **zero-initialised**, so at step zero $\gamma = \beta = 0$ and the whole modulation is exactly the identity. Conditioning is something the operator learns to use, not something imposed on it before it has learned anything.

Why FiLM comes after the normalisation and not before. Because a normalisation removes whatever a modulation just applied, to the extent that the modulation is constant within a group. Applying the same (γ, β) on either side of `GroupNorm` and measuring the change against no modulation at all:

Groups	Channels per group	Relative change
modulation before the norm		
16	1	3×10^{-6}
8	2	0.32
4	4	0.36
1	16	0.39
modulation after the norm — what Poraquê does		
8	2	**1.16**

With one channel per group the modulation is annihilated exactly: the normalisation restores the mean and variance the affine map had just changed, and only its *sign* survives. At the shipped eight groups of two, roughly a third of it survives, as within-group contrast. Placed after the norm it acts in full. The ordering is not a stylistic preference — it is the difference between conditioning that works and conditioning that is mostly undone one line later.

The residual comes last. $v^{(\ell)} = v^{(\ell-1)} + \sigma(\cdot)$ means each layer contributes a correction rather than replacing the field, which is what makes a deeper stack trainable, and it is why the identity is reachable: with FiLM at its initialisation and σ small, a fresh block is close to a no-op.

> **Note**
> **The activation between Fourier layers is `silu` (or `gelu`), and it is already there.** σ in Eq. (3.14) *is* `model.activation`: one application per block, L in total. What differs from the canonical FNO is not whether the nonlinearity is present but where it sits — after the normalisation and the conditioning rather than immediately after the spectral sum, so that it acts on a signal of controlled scale that already knows which cell it belongs to. Applying it first would normalise the activation's output instead of its input, which is the arrangement normalisation exists to avoid.
> The σ in the read-out, Eq. (3.15), is the same one, so the stack contains $L+1$ nonlinearities and `activation` governs all of them: at the default `silu`, every one of them is SiLU.

3.8.3 Reading the cost off the indices

Each setting appears in exactly one place, and that placement *is* the scaling:

C **is the range of *both* sums in Eq. (3.10).** It is the input index i and the output index o of the same contraction, so the weight tensor R carries C^2 of them and the cost is **quadratic**. Nothing else in the architecture has this property.

$m_1 m_2 m_3$ **are free indices on R, not summed.** Every retained coefficient gets its own $C \times C$ block, so the count is **cubic** in `modes` — R has shape $(4, C, C, m_1, m_2, m_3)$, which is Eq. (3.10) read as an array.

L **counts repetitions of the block, not terms in a sum.** Equations (3.9)–(3.14) are applied L times with independent weights, so depth is **linear** — and it composes the learned convolutions, which is how it buys range that `modes` would charge cubically for.

C_P **appears only in Eqs. (3.15)–(3.16), outside the loop.** It is the range of one sum, evaluated once, so it is **linear and paid once** — $CC_P + C_P C_{\text{out}}$ weights in total.

$N_1 N_2 N_3$ **appears nowhere.** No index range above refers to the grid. That is Section 3.4 restated: $\mathbf{r}$ and $\mathbf{k}$ are arguments, not indices, and `resolution` enters only through Eq. (3.17), where it can *cap* m_i but never enlarge the weight tensor.

Note

σ in Eq. (3.14) is `model.activation` — by default `silu`, optionally one of the per-channel learnable Kolmogorov–Arnold variants — and so is the read-out's nonlinearity in Eq. (3.15). One key, $L + 1$ applications.
A per-channel variant is sized by the layer it sits in: C learned functions inside a Fourier block, C_P in the read-out.
`model.projection_activation` overrides the read-out alone. Its default is `null`, meaning "whatever `activation` is", which is the only thing a reader should have to know. It exists because the read-out has not always followed: it was a fixed GELU until 2026-08-28, so one key then governed L nonlinearities out of $L + 1$ and the exception appeared in no config. A checkpoint written before the change records no read-out nonlinearity, and is reloaded with GELU on exactly that evidence — which is a statement about old files, not a default for new ones.

3.8.4 What each one buys

`width` — how much information travels through. The number of channels the field is carried in between the lifting layer and the read-out, and therefore how many distinct features of the density the operator can hold at once. It is the operator's representational capacity.

`modes` — how far it travels in space. Mode m on an axis of length ℓ_i is a wavelength ℓ_i/m, so `modes` fixes both the finest feature the learned convolution can resolve and the longest range over which it can correlate. It is a *capacity limit*, not a requirement: on a grid too coarse to supply that many, fewer are used automatically, which is what lets one set of weights serve materials on different grids (Section 3.4).

`n_layers` — how many times the cycle is applied. Depth composes the learned convolutions. Two layers of M modes reach further than one layer of M modes, because the second acts on a field the first has already delocalised — so depth buys effective range at *linear* cost, which `modes` does not.

`projection_channels` — how the result is read out. The hidden width of the two-convolution head that maps the C channels back to the physical output. It appears once, at the end, never inside a Fourier layer, and is purely pointwise.

`resolution` — what the operator is shown. Not an architecture parameter. It is the longest grid axis after the spectral downsampling of Section 4.3, applied once when the cache is built, and it fixes the band of physics present in the data rather than the band the operator acts on.

3.8.5 Where the parameters are

The spectral tensor is the model. Counted on the shipped `platinum_W16-M8-L3` itself — its `ext2chg` half, $C = 16$, $M = 8$, $L = 3$, $C_P = 64$, and $C_{\text{in}} = 1$, $C_{\text{out}} = 2$ because the platinum data is spin-polarised and the density it predicts is (ρ, m) — with a complex weight counted as two parameters, which is what `FNO3d.n_parameters` does and therefore what the training log prints:

Component	Where	Parameters	Share
Spectral R	Eq. (3.10)	3 145 728	99.79%
FiLM	Eq. (3.13)	3 168	0.10%
Cell encoder	γ, β inputs	1 312	0.04%
Read-out	Eqs. (3.15)–(3.16)	1 218	0.04%
Pointwise W	Eq. (3.12)	816	0.03%
GroupNorm	Eq. (3.12)	96	0.00%
Lifting	Eq. (3.8)	80	0.00%
Total		3 152 418	

One term in one equation is 99.79% of the model. Cell conditioning and normalisation together are 0.15%, and the read-out — the whole of `projection_channels` — is 0.04%.

The second output channel is almost free, and the table says why: C_{out} appears only in Eq. (3.16), so predicting m alongside ρ costs $C_P + 1 = 65$ parameters, or 0.002%. Nothing in the spectral tensor knows how many fields it is carrying.

Each row is its equation's index ranges multiplied out. Note the leading 2 on the spectral row: R is a *complex* tensor, and a complex weight is two real parameters.

$$\text{spectral}: \ \mathbf{2} \times L \times 4 \times C^2 \times m_1 m_2 m_3 \tag{3.19}$$

$$\text{pointwise}: \ L \times (C^2 + C) \tag{3.20}$$

$$\text{lifting}: \ C\,(C_{\text{in}} + 3) + C \tag{3.21}$$

$$\text{read-out}: \ CC_P + C_P + C_P\,C_{\text{out}} + C_{\text{out}} \tag{3.22}$$

Only the first matters, and it grows as $C^2 M^3 L$ — the C^2 from the two summed channel indices, the M^3 from the three free mode indices, the L from the repetition.

Note

Why the training log prints roughly twice what the spectral formula gives. For the shipped model ($C = 16$, $M = 8$, $L = 3$),

$$4 \times C^2 \times M^3 \times L = 4 \times 256 \times 512 \times 3 = 1\,572\,864,$$

and that is right — it is the number of *complex* weights in R, exactly what the three "spectral tensors" hold. Two things stand between it and the 3 152 418 the log reports:

1. **a factor of two**, because `FNO3d.n_parameters` counts a complex weight as the two real numbers it is: 1 572 864 → 3 145 728;
2. **everything that is not** R: 6 690 parameters — the FiLM projections ($3 \times 1\,056$), the cell encoder (1 312), the read-out (1 218), the pointwise convolutions (3×272), GroupNorm (3×32) and the lift (80).

3 145 728 + 6 690 = 3 152 418, which is the table above. A one-channel model would report 65 fewer: its last convolution is $C_P C_{\text{out}} + C_{\text{out}} = 65$ rather than 130.

3.8.6 Cost is not the parameter count

Doubling each setting in turn, at the shipped configuration on a 32^3 grid (CPU, forward and backward, four threads):

Change	Parameters	Time
`width` $16 \to 32$	3.99×	2.89×
`modes` $8 \to 12$	3.37×	1.32×
`n_layers` $3 \to 6$	2.00×	1.97×
`projection_channels` $64 \to 128$	1.00×	1.31×

The two rows where the columns disagree are the ones worth remembering.

`modes` is *expensive in memory and cheap in time*: $(12/8)^3 = 3.375$ more weights, but the FFT is unchanged and only the contraction over retained coefficients grows, which is not the dominant term at this size. It is the knob that runs a model out of memory long before it runs it out of wall clock.

`projection_channels` is the reverse — *free in parameters and not free in time*. It adds 1 152 weights, 0.07% of the model, because it is one pointwise map at the end; but that map is evaluated at every one of the N^3 voxels, so its cost scales with the grid rather than with its own size. Read the parameter count alone and it looks like nothing, which is why it is often the best place to spend a budget that `width` would otherwise consume quadratically.

3.8.7 Resolution

Resolution is the only one of the five that changes the *data*. At the shipped architecture:

Grid	Voxels	Time
16^3	0.12×	0.33×
24^3	0.42×	0.68×
32^3	1.00×	1.00×
48^3	3.38×	2.36×
64^3	8.00×	6.21×

Time approaches the $N^3 \log N$ of the transforms as the grid grows, and falls short of it below 32^3 where fixed per-layer overhead dominates. Memory is the harder limit: activations are CN^3 per layer and scale exactly, with no overhead to hide behind.

Caveat

`resolution` and `modes` are not independent. An N^3 grid supplies $(N/2,\ N/2,\ N/2+1)$ modes — the last axis is the halved one of `rfftn` — so at `resolution: 32` any `modes` above 16 allocates weights that no sample can reach: allocated, saved into the checkpoint, never trained. The shipped $M = 8$ at $N = 32$ uses half the available band, which leaves room to raise one without the other.

Note

None of the four architecture parameters refers to N. That is the point of Section 3.4: the same weights are applied to a 24^3 and a 120^3 sample, and `resolution` can be changed without invalidating a checkpoint's shape. What it does invalidate is the *comparison* — a model trained at 32^3 has seen a narrower band of physics than one trained at 64^3, whatever their reported errors say.

3.8.8 Choosing them

In one clause each: `width` is how much information travels through; `modes` is how far it travels in space; `n_layers` composes the whole cycle; and `projection_channels` is only how the result

is read out.

A budget is usually best spent in that reverse order:

1. `projection_channels`, first, because it is nearly free in parameters — but judge it on the wall clock, not on the model size.
2. `n_layers`, when the operator needs more *range*: it buys that linearly, where `modes` buys it cubically.
3. `width`, when it needs more *capacity*, and pay the square.
4. `modes`, last and deliberately, after checking it against `resolution`.

CHAPTER 4

Implementation

4.1 The shared-grid data model

The organising invariant of `poraque.fields` is that, for one material, every scalar field lives on *one* `FieldGrid` instance and is serialised in *one* format. Everything else follows.

Class	File	Content
`ExternalPotential`	`EXTCAR`	V_{ext}, eV
`ChargeDensity`	`CHGCAR`	ρ, $e/\text{Å}^3$
`KineticEnergyDensity`	`TAUCAR`	τ, $\text{eV}/\text{Å}^3$

All three derive from `ScalarField`, which owns the `CHGCAR`-format reader and writer. One code path therefore handles every field, and reading a field with a supplied grid *imposes* that grid — a mismatch raises rather than passing silently, since a misaligned pair would train the operator on nonsense.

Implementation

`ChargeDensity` and `KineticEnergyDensity` declare `volume_scaled = True`: VASP stores $\rho\Omega$ and $\tau\Omega$, so the conversion happens transparently on read and write, and `integrate()` returns the electron count directly. Verified against the reference data: 297.0000 electrons for 27 platinum atoms at $Z^{\text{val}} = 11$. `ExternalPotential` declares it `False`, matching `LOCPOT`.

4.2 Grid construction

The grid may be obtained in three ways, in decreasing order of reliability:

1. `FieldGrid.from_file(path)` — adopt the grid of an existing volumetric file. The only way to guarantee point-by-point comparability with a reference calculation, and the recommended default.
2. explicit `NGXF/NGYF/NGZF` tags from the input file;
3. `FieldGrid.from_encut` — derived from the plane-wave cutoff.

The cutoff rule uses $G_{\text{cut}} = \sqrt{E_{\text{cut}}/(\hbar^2/2m_e)}$ and hence $n_i^{\max} = \lfloor G_{\text{cut}}|\mathbf{a}_i|/2\pi \rfloor$, rounded up to even, FFT-friendly sizes.

Caveat

The cutoff-derived shape is an *estimate*. VASP's exact rounding is version- and setting-dependent: for the platinum data set the rule yields 64^3 where VASP used 128^3, and on a strained cell VASP chose 120 on one axis where the rule gives 128. Every result in this guide reads the grid from the files, so no conclusion depends on the heuristic — but it should not be relied on when a match is required.

4.3 Spectral resampling

Changing the resolution of a plane-wave field is a *basis-truncation* problem, not an interpolation problem. The field is a finite Fourier series, so restricting it to a coarser grid means keeping the coefficients the coarse grid can represent:

$$f(\mathbf{r}) = \sum_{\mathbf{G}} c_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}} \longrightarrow \sum_{|\mathbf{G}|\leq \mathbf{G}_{\max}^{\text{coarse}}} c_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}}. \tag{4.1}$$

Nothing is approximated: the result is the exact band-limited projection, it remains exactly periodic, and because the $\mathbf{G} = 0$ coefficient is copied unchanged, the cell average — hence $\int \rho \, \mathrm{d}\mathbf{r}$ — is preserved to machine precision.

Implementation

Measured: a band-limited field downsampled $64^3 \to 32^3$ agrees with direct sampling to 1.3×10^{-15}; the mean is preserved to 5.6×10^{-17}; and the operation is idempotent under down–up–down to 6.7×10^{-16}. Implemented in `poraque/fields/resample.py`.

Caveat

Band-limiting a strictly positive field with sharp core peaks rings slightly (the Gibbs phenomenon), so a small number of voxels can go marginally negative — one point in 32768 for this data. It is an artefact of the truncation, not of the data, and it is why the data pipeline uses a sign-tolerant asinh normalisation rather than a logarithm.

4.4 Code-agnostic ingestion

Support for a new plane-wave code must not ripple through grids, fields, datasets and models. `poraque.fields.io` therefore defines a narrow contract, and a reader implements exactly four methods:

Method	Returns
`read_structure`	`Structure` (Å, fractional, species-grouped)
`read_parameters`	`CalculationParameters`, cutoff in **eV**
`read_pseudopotentials`	{element: `PseudopotentialInfo`}
`read_field` / `write_field`	`ScalarField`

Units are normalised at the boundary: a Quantum ESPRESSO reader converts its Rydberg `ecutwfc` in `read_parameters`, and nothing downstream needs to know. Field kinds are referred to by neutral names (`external`, `density`, `kinetic`) rather than by filename, since `CHGCAR` has no meaning outside VASP.

`VaspReader` is the reference implementation. `EspressoReader` and `GpawReader` are scaffolded, each documenting the specific traps: for Quantum ESPRESSO the Rydberg cutoff, the `ibrav` cell construction, the `alat/crystal` coordinate units, and the absence of a volume pre-factor; for GPAW the coarse-versus-fine grid distinction and the pseudo-versus-all-electron density ambiguity, which would be a silent physics error if mixed across a data set.

4.5 Hardware acceleration

`resolve_device` prefers CUDA, then Apple Metal (MPS), then CPU. An explicit request for an absent backend warns and falls back rather than raising, so a configuration written on a workstation still runs on a laptop.

Apple Metal required three genuine workarounds, all verified by regression tests:

No float64, no `linalg.det`. Both are used for the cell metric. Since a 3×3 inverse and determinant are $\mathcal{O}(1)$ work beside an FFT, they are evaluated *on the host in double precision* and only the result is moved to the device — which also keeps the extra accuracy on every backend. Note the ordering: a tensor must be moved to the host *before* widening, since casting an MPS tensor to float64 in place raises.

No complex `einsum`. The spectral contraction lowers to an `mps.gather` over complex values, which Metal rejects by aborting the process. Splitting the product into real arithmetic, $(a+ib)(c+id) = (ac-bd)+i(ad+bc)$, uses only real contractions and is numerically identical.

Strided complex views are silently wrong. Taking `.real` of a *non-contiguous* complex tensor yields a strided real view over which MPS miscomputes `einsum` — with no error and results 40–90 % off. The operands here are precisely such views (high-frequency corners such as `x_ft[..., nx-m1:, ny-m2:, :m3]`). Materialising them first reduces the error to $\sim 10^{-7}$.

Gradient clipping also required replacement: `clip_grad_norm_` calls `linalg.vector_norm`, unsupported for complex dtypes on MPS, and the spectral weights are complex. Viewing a complex tensor as its stacked $(\mathrm{Re}, \mathrm{Im})$ representation is exact — $\sum_k |z_k|^2 = \sum_k (a_k^2 + b_k^2)$ — so the clip is identical on every backend.

Grid	CPU	MPS	speed-up
32^3	42.0 ms	19.2 ms	2.19×
48^3	124.3 ms	38.3 ms	3.24×
64^3	262.2 ms	45.9 ms	5.71×

Forward pass, five-run average, synchronised. CPU and MPS agree to $\sim 10^{-6}$ relative on a single forward pass with identical weights.

4.6 A C kernel for CPU inference

The accelerator path above is the answer when there is an accelerator. On CPU the picture is different, and profiling says where:

Grid	`rfftn`	contract	`irfftn`	total	contract
32^3	2.1 ms	16.7 ms	2.8 ms	22.6 ms	74 %
64^3	16.7 ms	16.7 ms	22.5 ms	57.9 ms	29 %
96^3	62.0 ms	16.7 ms	80.8 ms	164.2 ms	10 %

One spectral layer, width 32, modes 12. The FFTs scale as $N^3 \log N$; the contraction does not scale with the grid at all, because its cost is fixed by the retained mode count — a model hyper-parameter. So at the 32^3 resolution a training cache is normally built at, *the contraction is the layer*.

And `torch.einsum` runs it at roughly 3.4 GFLOP/s. The pattern `bixyz,ioxyz->boxyz` reduces to a batched product over a layout whose reduction axis is strided by the whole mode block. Reordering the loops so the flattened mode index is innermost — contiguous in all three operands — is the whole optimisation, and it is not expressible through `einsum`:

```
for b, o:                        /* accumulator row stays in L1 */
    zero(acc)
    for i:
        for m in 0..M:           /* contiguous in x, w and acc */
            acc[m] += x[b][i][m] * w[i][o][m]
```

Path	one contraction	GFLOP/s
`torch.einsum` (4 threads)	4.53 ms	2.6
C, serial	0.61 ms	23.1
C, 4 pthreads	0.20 ms	67.3

Batch 1 — the shape every prediction has. End to end that is 2–3.4× on whole-model inference at 24^3–32^3, tapering to ∼1× at 96^3 where the FFTs dominate again.

4.6.1 Why pthreads, and why not MPI

The kernel's arithmetic intensity is **1 FLOP per byte**: at batch 1 every weight element is read exactly once for one complex multiply–add. It is memory-bound, not compute-bound, which decides both questions.

Threading is POSIX threads rather than OpenMP because PyTorch already links an OpenMP runtime, and a second one in the same process is what libomp reports as `OMP: Error #15 ... can degrade performance or cause incorrect results` — whose only documented workaround its own documentation calls unsafe. pthreads have no runtime to collide with. Threads buy aggregate memory bandwidth, so the useful count saturates well below the core count: measured, at four of eight.

MPI was implemented and measured, then removed. Splitting the same output rows across ranks is correct to 2.7×10^{-7} and reaches 0.26 ms on four ranks — the *same* bandwidth ceiling as threads, while costing a process and a full model replica per rank plus a gather on every one of the sixteen contractions in a forward pass. The operands already share an address space; moving them between processes spends the one resource that is scarce. Across nodes it is hopeless regardless: the kernel is 0.2 ms, below the latency floor of any interconnect. Rank-level parallelism belongs one level up, over structures, where it is embarrassingly parallel and communicates one scalar per structure.

4.6.2 How it is loaded

Plain C, no Python C API and no numpy headers, called through `ctypes`. That keeps the extension out of the build system entirely: the wheel is pure Python, and the shared library is compiled on first use into `~/.cache/poraque` and reused. Every failure path — no compiler, rejected flags, an unwritable cache, an unsupported dtype or device, `PORAQUE_C_BACKEND=0` — falls back to `torch.einsum`. It is an optimisation and never a dependency.

> **Caveat**
> The kernel writes into a plain tensor and records nothing on the autograd tape, so it is gated on `torch.is_grad_enabled()`. Using it while a graph is being built would leave the spectral weights with no gradient — a model that trains to a constant, with no error anywhere and a loss curve that merely looks disappointing.

4.7 Numerical precision

The operator carries **complex** parameters — the mode multipliers of `SpectralConv3d` — alongside the real weights of its pointwise layers, and converting between precisions has to move both together. Neither PyTorch idiom does:

`model.double()` converts only tensors for which `is_floating_point()` is true. A complex64 spectral weight is not one, so it is left behind and the next forward pass dies with `expected scalar type ComplexDouble but found ComplexFloat`.

`model.to(torch.float64)` is worse: it casts complex64 to *float64*, discarding the imaginary part of every Fourier multiplier. No error, and a model that has silently lost half of its spectral parameters.

`set_precision` maps real dtypes to the real target and complex dtypes to the matching complex target, which is the only conversion that leaves the operator meaning what it meant. It is reached from `model.precision`, and is separate from `data.precision`, which governs only how the volumetric fields are held in memory.

4.8 Configuration

A training run is defined by a YAML file with four sections — `data`, `model`, `training`, `output` — mirroring the four things a run needs; Chapter 9 walks the pipeline end to end. Command-line flags override individual entries, so one committed config can be swept from the shell without being edited:

```
poraque-train --config configs/train.yaml --epochs 500
```

Unknown keys are rejected rather than ignored: a typo such as `learn_rate` would otherwise be silently dropped, and the run would quietly use the default while appearing to be described by the file. The *resolved* configuration is written beside the results, recording what actually ran, overrides included.

CHAPTER 5

The VASP interface

Poraquê computes the external ionic potential natively. Reference `EXTCAR` and `TAUCAR` files were used during development to *validate* that reconstruction. They are not required to run Poraquê — nothing in the pipeline reads them — and the reference calculations themselves are ordinary VASP 6.6.1 output in every other respect.

5.1 `EXTCAR`: the local pseudopotential

5.1.1 What is written

`main.F` calls `POTION` to build the potential in reciprocal space, transforms it to real space, and writes it with `OUTPOS` + `OUTPOT`. The choice of `OUTPOT` rather than `OUTCHG` is what makes `EXTCAR` a *potential* file in the `LOCPOT` sense: **no multiplication by** Ω. The source comment states the conventions directly: values in eV, no volume scaling, $\mathbf{G} = 0$ set to zero, a single spin-independent block, no PAW one-centre terms and no non-local projector contribution.

5.1.2 The formula

`POTION` (`pot.F`) contains, per species and per $\mathbf{G}$ vector:

```
ARGSC  = NPSPTS/P(NT)%PSGMAX
PSGMA2 = P(NT)%PSGMAX - P(NT)%PSGMAX/NPSPTS
ZZ     = -4*PI*P(NT)%ZVALF*FELECT

G = SQRT(GX**2+GY**2+GZ**2)*2*PI
IF ( (G /= 0) .AND. (G < PSGMA2) ) THEN
   I    = INT(G*ARGSC)+1
   REM  = G - P(NT)%PSP(I,1)
   VPST = PSP(I,2)+REM*(PSP(I,3)+REM*(PSP(I,4)+REM*PSP(I,5)))
   CVPS(N) = CVPS(N) + ( VPST + ZZ/G**2 ) / LATT_CUR%OMEGA * CSTRF(N,NT)
ELSE
   CVPS(N) = 0._q
ENDIF
```

which is

$$\boxed{V_{\text{ext}}(\mathbf{G}) = \frac{1}{\Omega}\sum_s S_s(\mathbf{G})\left[v^s_{\text{short}}(G) \;-\; \frac{4\pi Z_s^{\text{val}} e^2}{G^2}\right], \qquad V_{\text{ext}}(\mathbf{0}) = 0.} \tag{5.1}$$

Comparing with Eq. (2.6): the second term is *exactly* the bare-Coulomb form factor $f_s = 1$. The long-range physics was never the problem. The first term, v_{short}, is a cubic spline through a *tabulated* function read from the `POTCAR` — and that is the entire discrepancy.

Two further details are easy to miss and both are observable:

- **Hard truncation.** For $G \geq$ PSGMA2 the *whole* contribution is zeroed, Coulomb tail included; VASP does not extrapolate the table. For the platinum potential PSGMAX $= 75.6$ Å^{-1} while a 128^3 grid reaches $|\mathbf{G}|_{\max} = 106$ Å^{-1}, so the truncation is genuinely active and must be reproduced.
- **G carries the 2π.** LATT_CUR%B is the reciprocal lattice *without* 2π, so G is the ordinary angular wavevector in Å^{-1} — the same convention as FieldGrid.get_g_vectors.

5.1.3 The POTCAR table layout

The layout of the local part block is given by the reader in pseudo.F:

```
READ(10,'(1X,A1)') CSEL          ! the "local part" marker line
READ(10,*) P(NTYP)%PSGMAX        ! <-- FIRST number is PSGMAX, not ZVAL
ALLOCATE(P(NTYP)%PSP(NPSPTS,5))
READ(10,*) (P(NTYP)%PSP(I,2),I=1,NPSPTS)
DO I=1,NPSPTS
    P(NTYP)%PSP(I,1)=(P(NTYP)%PSGMAX/NPSPTS)*(I-1)
ENDDO
P(NTYP)%PSCORE=P(NTYP)%PSP(1,2)
CALL SPLCOF(P(NTYP)%PSP(1,1),NPSPTS,NPSPTS,0._q)
```

with NPSPTS $= 1000$. The mesh is therefore uniform,

$$q_i = \frac{\texttt{PSGMAX}}{1000}(i-1), \qquad i = 1, \ldots, 1000, \tag{5.2}$$

and the values are in eV · Å^3.

> **Note**
> The first number after the marker is PSGMAX, the table's maximum wavevector — *not* the valence charge. The two can look alike in some files, and the misreading was the secondary cause of the original discrepancy. Confirmed against the platinum POTCAR: the value is 75.5890395431569, the block holds 1001 numbers (= PSGMAX + exactly 1000 samples), and ZVAL is 11.

PSCORE $= v_{\text{short}}(q \to 0) = 105.89$ eV · Å^3 for platinum feeds the PSCENC energy correction and does not enter the potential, since $V_{\text{ext}}(\mathbf{G} = 0)$ is zeroed.

5.1.4 Result

Implementing Eq. (5.1) verbatim — natural cubic spline on the PSGMAX mesh, analytic Coulomb tail, $\mathbf{G} = 0$ zeroed, truncation at PSGMA2, structure factors per species — gives:

Structure	grid	Gaussian model	**tabulated**
struct_000	128^3	0.1338	$\mathbf{2.13 \times 10^{-5}}$
struct_001	120×128×128	0.1303	$\mathbf{5.95 \times 10^{-5}}$
struct_002	128×120×128	0.1285	$\mathbf{5.75 \times 10^{-5}}$

(relative L^2 against the reference EXTCAR; Pearson $r = 1.000000$ in all three, MAE 0.34 meV.) The improvement is a factor of roughly 6000.

> **Caveat**
> The residual $\sim 6 \times 10^{-5}$ is the spline boundary condition: VASP calls SPLCOF(PSP(1,1),NPSPTS,NPSPTS,0._q) whereas the Python implementation uses a nat-

ural cubic spline. The difference is confined to the ends of the table and is worth about 0.6 meV. It is documented rather than chased.

5.1.5 Why an analytic form factor cannot work

For a single-species cell Eq. (5.1) can be inverted to recover the empirical form factor directly from a reference file,

$$f(\mathbf{G}) = -\frac{\Omega\, G^2\, V_{\text{ext}}^{\text{ref}}(\mathbf{G})}{4\pi e^2 Z^{\text{val}} S(\mathbf{G})}. \tag{5.3}$$

Doing so for platinum shows $f(G)$ decaying far more slowly than a Gaussian at low G and then *oscillating about zero* at large G — the signature of a repulsive pseudopotential core. No monotonic $e^{-G^2\sigma^2/2}$ can reproduce that, which is why fitting σ saturates near a relative L^2 of 0.13 regardless of the value chosen. The best-fit width came out at $\sigma^* = 0.374$ Å on all three structures independently, a consistency that identified the limitation as structural rather than accidental.

5.2 TAUCAR: the kinetic energy density

5.2.1 Definition

The source comment gives `tau_s(r) = hbar^2/2m sum_nk w_k f_nk |grad psi_nk,s(r)|^2`, i.e. the positive-definite gradient form of Eq. (1.9), and *not* the Laplacian form. `TAU_PW` in `metagga.F` evaluates it per Cartesian direction:

```
DO I=1,NPL
   G1=WDES%IGX(I,NK)+WDES%VKPT(1,NK)
   GC=(G1*LATT_CUR%B(IDIR,1)+G2*LATT_CUR%B(IDIR,2)+G3*LATT_CUR%B(IDIR,3))
   CFA(I)=GC*W%CPTWFP(...)*CITPI          ! i*2pi*(G+k)_IDIR * c_G
ENDDO
CALL FFTWAV(NPL,WDES%NINDPW(1,NK),CFAR(1),CFA(1),GRID)
DO I=1,GRID%RL%NP
   CKIN(I)=CKIN(I)+HSQDTM*REAL(CONJG(CFBR(I))*CFAR(I))*WEIGHT
ENDDO
```

The gradient is applied *spectrally* as $i(\mathbf{G}+\mathbf{k})$ on the plane-wave coefficients, inverse-transformed to real space, and accumulated as $|\partial_\alpha \psi|^2$ weighted by $w_k f_{nk}$, with `HSQDTM` $= \hbar^2/2m_e$.

5.2.2 Storage

`CTAUC = CTAUC*LATT_CUR%OMEGA` followed by `OUTCHG` means `TAUCAR` stores $\tau\Omega$, exactly as `CHGCAR` stores $\rho\Omega$, so that $\sum_{\mathbf{r}}$ `TAUCAR`$(\mathbf{r})/(N_1 N_2 N_3)$ is the plane-wave kinetic energy of the cell in eV. For `ISPIN=2` the two blocks are (total, magnetisation), matching the `CHGCAR` layout.

5.2.3 Verdict

Poraquê's handling of `TAUCAR` required no change: the positive-definite definition, the Ω factor, the units, the shared grid and the plane-wave-only content all matched already. That τ is the pseudo quantity also explains an earlier observation — the bound (1.10) was found to hold at essentially every grid point (one violation in 32768, at the single voxel where spectral downsampling had rung τ slightly negative), because τ and ρ are pseudised *consistently*. The bound is strictly guaranteed only for all-electron quantities, so this was worth measuring rather than assuming.

5.3 What is absent from EXTCAR

Worth stating explicitly, because it bounds what the first learned map can be asked to do:

- no PAW one-centre terms — EXTCAR is a pseudo quantity;
- no non-local projectors — not local functions of $\mathbf{r}$, so they cannot appear in a volumetric file even in principle;
- no Hartree and no exchange–correlation, unlike LOCPOT which is $v_{\mathrm{H}} + V_{\mathrm{ext}}$. EXTCAR is the bare ionic term, which is precisely the right input for the $V_{\mathrm{ext}} \mapsto \rho$ map;
- **no Ewald or ion–ion contribution.** That energy is a scalar (PSCENC, TEWEN), not a field; none is missing, because none belongs in this file.

Caveat

Two paths remain untested. The multi-species sum in Eq. (5.1) uses a per-species PSGMAX and is implemented as such, but the present data set is pure platinum. And ISPIN=2 writes a second block in both CHGCAR and TAUCAR which the reader currently ignores — correct for these ISPIN=1 runs, a silent truncation otherwise.

5.4 When the pseudopotentials are absent

Everything above assumes a POTCAR is at hand. A large and growing class of training data does not have one: the Materials Project publishes converged charge densities for a substantial fraction of its materials and ships a density and nothing else — no POSCAR, no INCAR, no POTCAR, no OUTCAR, no TAUCAR. Since V_{ext} is the *input* of the first learned map, its construction there deserves stating precisely.

5.4.1 What the density alone can supply

Two of the three ingredients are recoverable from the file itself.

The geometry. A CHGCAR opens with a complete POSCAR block, so the lattice, the species and the fractional coordinates come from the density. Nothing is inferred.

The valence charges. A pseudopotential density integrates to the valence electron count of its own cell,

$$\sum_s n_s^{(m)} Z_s^{\mathrm{val}} \;=\; N_e^{(m)} \;=\; \frac{1}{N_1 N_2 N_3} \sum_{\mathbf{r}} \mathtt{CHGCAR}^{(m)}(\mathbf{r}), \tag{5.4}$$

one linear equation per material m in the per-element charges, with the stoichiometry $n_s^{(m)}$ read from the header. A chemical space with more distinct compositions than elements over-determines the system, and least squares recovers the charges the POTCAR would have stated. Only as many densities are read as the system needs — candidates are taken smallest-file-first and accepted only while they add rank — so a hundred-material space costs a handful of small reads rather than a full pass.

On the Ag–Pt–Pt space this returns 11, 11 and 10 with a residual of zero to machine precision, from three elemental cells: exactly the ZVAL of the three PAW_PBE potentials, read off the data.

5.4.2 What it cannot: the short-range table

The third ingredient is $v^s_{\text{short}}(G)$, and no density determines it. Equation (5.1) splits V_{ext} into a Coulomb tail fixed by Z^{val} — now known — and a short-range remainder that is a property of the *pseudopotential*, not of the system it was used on. Two routes remain.

Supply the library. `data.potcar_dir` points at a `POTCAR` library, one entry per species as VASP distributes them. The archive then supplies the structure and the library the pseudopotentials, which together are everything Eq. (5.1) requires: the construction is the same one, and so is the accuracy, a relative 2×10^{-5}.

This is not merely the better option, it is the one that keeps the input field a *single physical quantity* across a mixed dataset. Training on archive densities beside local runs otherwise means the operator sees two different definitions of V_{ext} wearing one name.

Fall back to the model form factor. Absent a library, the Gaussian pseudo-ion model is used — $f_s(G) = e^{-G^2\sigma_s^2/2}$ in place of the tabulated v^s_{short} — with σ defaulting to 0.5 Å, since no `RCORE` is available to derive one from either. Measured on the Ag–Pt–Pt densities, the two constructions differ from *each other* by

$$\frac{\|V^{\text{tab}}_{\text{ext}} - V^{\text{gauss}}_{\text{ext}}\|_2}{\|V^{\text{tab}}_{\text{ext}}\|_2} = 0.38, \tag{5.5}$$

comparable with the 0.13 that the best-fit σ reaches against a reference `EXTCAR`, and far above any accuracy the learned map is expected to reach. The inversion argument above settles why this cannot be fitted away: the empirical $f(G)$ oscillates about zero at large G and no monotonic Gaussian reproduces that.

5.4.3 What the fallback licenses

A model trained without the library learns

$$V^{\text{gauss}}_{\text{ext}} \longmapsto \rho^{\text{DFT}}, \tag{5.6}$$

which is a well-posed and self-consistent learning problem — the same model potential is constructed at inference time, so the map is single-valued and the operator is never asked to reconcile two inputs. It is simply *not the Hohenberg–Kohn map of VASP's* V_{ext}. Its accuracy is therefore not comparable with a model trained on tabulated potentials, and any table putting the two side by side has to say which construction each used.

Degradation is per species, not per dataset: a library missing one element of five falls back for the structures containing that element and keeps the exact potential for all the others. The run states the construction it used for each source, and the cache directory name records it, so the two can never be mixed in one cache by accident.

CHAPTER 6

Results

6.1 The data set

Three platinum structures, each a $3 \times 3 \times 3$ supercell of the face-centred cubic primitive cell (27 atoms, $Z^{\mathrm{val}} = 11$, so 297 valence electrons), computed with VASP 6.2.0 at $E_{\mathrm{cut}} = 450\,\mathrm{eV}$, `PREC=Accurate`, and a $5 \times 5 \times 5$ Γ-centred k-mesh.

Structure	Geometry	Ω [Å^3]	Native grid
struct_000	pristine fcc	484.53	$128 \times 128 \times 128$
struct_001	rattled + strained	477.65	$120 \times 128 \times 128$
struct_002	rattled + strained	476.92	$128 \times 120 \times 128$

The three native grids *differ*, which is not an inconvenience but the central test: it is exactly the situation Section 3.4 was designed for, and it arises automatically because the cell shape enters VASP's grid selection.

> **Implementation**
> Fields are reduced to a longest axis of 32 points by the spectral truncation of Section 4.3, giving 32^3, $30 \times 32 \times 32$ and $32 \times 30 \times 32$. The shape difference *survives* the reduction — had a fixed target shape been used instead, the ragged-grid capability would have gone untested.

6.2 Verification before learning

Nothing was trained until the field machinery was checked against quantities known independently.

Check	Expected	Measured
Poisson residual, $\nabla^2 V_{\mathrm{ext}}$ vs $4\pi e^2 n_{\mathrm{ion}}$	0	4.5×10^{-12}
Translation covariance	exact	2.1×10^{-14}
Linearity in Z^{val}	exact	0.0
$\int \rho \, \mathrm{d}\mathbf{r}$ from `CHGCAR`	297	297.0000
Spectral resampling, band-limited	exact	1.3×10^{-15}
Resampling, mean preserved	exact	5.6×10^{-17}
Madelung constant (rock salt)	1.7475645946	1.74756459
`EXTCAR` vs VASP, tabulated	—	2.1×10^{-5} rel. L^2

The electron count is the sharpest of these: it exercises the `CHGCAR` parser, the Ω convention and the grid metric simultaneously, and returns the exact integer.

6.3 Model 1: $V_{\text{ext}} \mapsto \rho$

Leave-one-out over the three materials; metrics on the held-out material in physical units ($e/\text{Å}^3$).

Held out	MAE	RMSE	rel. L^2	R^2
struct_000	0.01667	0.03097	0.0318	0.9983
struct_001	0.02433	0.03984	0.0405	0.9973
struct_002	0.02423	0.04112	0.0418	0.9971
mean	**0.02174**	**0.03731**	**0.0380**	**0.9976**
baseline: mean density	0.5199	0.7589	0.7779	0.0000

The learned map is a factor of 20 better than predicting a constant density. The electron count, which nothing in the objective constrains, is recovered to 1.1–2.2 % — precisely the single global degree of freedom that the charge-normalisation head of Section 7.1.3 would fix exactly and for free.

6.4 Model 2: $\rho \mapsto \tau$

Same protocol, in $\text{eV}/\text{Å}^3$, with the Pauli head of Eq. (7.2) enabled. The analytic orbital-free functionals of Chapter 2 are evaluated on the same input as baselines.

Predictor	MAE	RMSE	rel. L^2	R^2
FNO (learned)	**0.6600**	**1.2002**	**0.0620**	**0.9924**
von Weizsäcker	8.99	14.22	0.7367	0.0150
Thomas–Fermi	9.38	25.99	1.3454	−2.2908
TF + vW/9	9.35	26.18	1.3554	−2.3395

The learned functional is better than the best classical approximation by a factor of about 12 in relative L^2.

> **Note**
> Thomas–Fermi attains a *negative* coefficient of determination: as a pointwise predictor of τ it is worse than the constant mean. This is not a defect of the implementation but the expected failure of a uniform-electron-gas limit applied to a d-band metal, whose core and d-shell regions are about as far from uniform as matter gets. It is the gap that motivates a learned functional.

6.4.1 Integrated kinetic energy

The quantity that enters a total energy is $T_s = \int \tau \, \mathrm{d}\mathbf{r}$, not τ pointwise:

Held out	predicted [eV]	reference [eV]	error
struct_000	6277.16	6207.07	1.13 %
struct_001	6198.07	6214.80	0.27 %
struct_002	6180.76	6217.06	0.58 %

6.4.2 Constraint satisfaction

With the head enabled, $\tau \geq \tau_{\mathrm{vW}}[\rho]$ held at every one of the $\sim 3 \times 10^4$ grid points on all three folds, with minimum margins of 1.045, 0.950 and 1.024 eV/Å^3. The margins are the informative number: the model is not sitting on the bound but predicting a Pauli term comfortably above it.

The fitted scale s came out at 5.22, 5.02 and 5.01 eV/Å^3 across folds — consistent with the median τ_{P} measured directly from the data (4.81–5.21), which is a check that `fit_pauli_scale` estimates what it claims to.

Section 7.2 gives the matched comparison against an unconstrained backbone: the head improves every fold and every metric, by 18.3 % in mean relative L^2, while training 17.8 % *faster*.

6.5 The inference pipeline

Chaining both models turns a bare crystal geometry into predicted ρ and τ with no wavefunctions and no self-consistency cycle:

$$\{\texttt{POSCAR}, \texttt{INCAR}, \texttt{POTCAR}\} \xrightarrow{\text{analytic}} V_{\mathrm{ext}} \xrightarrow{\text{FNO 1}} \rho \xrightarrow{\text{FNO 2}} \tau. \tag{6.1}$$

Only the two field-to-field maps are learned; the first step is closed-form.

The importance of Chapter 5's correction shows up here most sharply. Evaluating on struct_002 with models that never saw it:

Field	Gaussian V_{ext}	**tabulated V_{ext}**
`EXTCAR`	0.6645	**0.0014**
`CHGCAR`	0.4482	**0.0647**
`TAUCAR`	0.2456	**0.1190**

(relative L^2 against the reference files.) The models were always *trained* on VASP's own `EXTCAR`; the error was entirely in the inference path, which had been feeding them a potential from a different distribution. Correcting the analytic step alone improved the predicted density sevenfold, with no retraining.

> **Implementation**
> All predicted fields are written in `CHGCAR` format. `-compare` additionally brings reference files onto the prediction grid by spectral truncation — the exact restriction — rather than failing on a shape mismatch.

6.6 Hardware

The full run — two tasks, three folds each, 200 epochs, spectral downsampling, figures and checkpoints — completes in roughly 20 minutes on Apple Silicon via MPS. Forward-pass throughput is given in Section 4.5; the advantage grows with grid size, from 2.2× at 32^3 to 5.7× at 64^3, because kernel-launch overhead dominates at small sizes while the FFTs dominate at large ones.

6.7 What these numbers do and do not support

Caveat

With three closely related structures of a single element, leave-one-out measures **interpolation between nearby geometries of platinum**. It demonstrates that the pipeline is correct, that the architecture has sufficient capacity, and that the learned kinetic functional beats the classical approximations on this system. It says **nothing** about transfer to other chemistry, other coordination environments, or other cutoffs. The training log carries this caveat in its own footer so that it travels with the numbers.

What *is* established independently of data-set size:

- the field machinery is verified against closed-form physics (Poisson, Madelung, electron count) to machine precision;
- the `EXTCAR` reconstruction matches a reference implementation to 2×10^{-5};
- one set of weights serves three different grid shapes, with CPU/accelerator agreement at 10^{-6};
- the von Weizsäcker bound is structural, verified at random initialisation and under adversarial optimisation;
- the classical functionals' failure on this system is quantified rather than asserted.

The obvious next step is more materials. The ingestion layer, the shape-bucketed sampler and the configuration system were all built for a heterogeneous data set; the binding constraint is now the data, not the code.

CHAPTER 7

Physics-informed operators

7.1 Constrain by construction, not by penalty

A soft penalty $\lambda\|\text{violation}\|^2$ trades accuracy against constraint satisfaction and fully achieves neither: the weight must be tuned, the constraint holds only approximately, and a badly scaled term degrades the model while looking principled. Where a constraint can be built into the *parameterisation* it should be. It then holds for every weight configuration, at initialisation and after any optimiser step, with nothing to tune and no inference cost.

Three constraints move into the architecture.

7.1.1 Positivity

Predict $\log \rho$ and exponentiate. The inverse transform returns $e^y - \epsilon$, so $\rho > -\epsilon$ *identically*.

7.1.2 Electron count

$$\rho_{\text{out}}(\mathbf{r}) = N_{\text{val}} \frac{\tilde{\rho}(\mathbf{r})}{\int \tilde{\rho}\, \mathrm{d}\mathbf{r}} \tag{7.1}$$

makes $\int \rho = N_{\text{val}}$ hold to machine precision, differentiably, at the cost of one reduction. Strictly better than a penalty on the same quantity.

7.1.3 The von Weizsäcker bound

Using the exact decomposition (1.13), write the network output as

$$\boxed{\tau_\theta = \tau_{\text{vW}}[\rho] \;+\; s \cdot \text{softplus}(f_\theta(\rho))} \tag{7.2}$$

so the network learns only the Pauli term. This buys two things at once:

1. **The bound becomes structural.** `softplus` is non-negative, so $\tau_\theta \geq \tau_{\text{vW}}$ identically. The inequality is enforced *non-strictly*, exactly as Hoffmann-Ostenhof states it: for a strongly negative output `softplus` underflows and the head returns $\tau = \tau_{\text{vW}}$ exactly — the correct one-orbital limit, which is *reachable* rather than merely approachable. A soft penalty can only approach it from above.

2. **The network stops re-deriving known physics.** On the platinum data τ_{vW} accounts for $\sim 31\,\%$ of τ; that fraction is now supplied analytically and exactly by a spectral gradient, leaving only the genuinely unknown remainder to learn.

> **Implementation**
>
> The scale s is fitted on the *training split only* — fitting it on the held-out material would

leak the answer — and is then optimised as $\log s$, which cannot change sign. $\tau_{\mathrm{vW}}[\rho]$ is a fixed function of the *input*, so it contributes no gradient to θ: it is a per-sample offset, not a second trainable branch.

Caveat

The bound is a theorem for all-electron densities; `CHGCAR` and `TAUCAR` are pseudo quantities, so it must be *measured* before the head is enabled on a new data set. `pauli_bound_violation` does this. On the present data it holds at every point of two structures and at all but one point of the third.

7.2 Measured effect of the head

Identical data, seed, architecture and epoch count; only the head differs.

Held out	baseline	**Pauli head**	change
struct_000	0.0534	**0.0363**	−32.0 %
struct_001	0.0840	**0.0762**	−9.3 %
struct_002	0.0902	**0.0734**	−18.6 %
mean	0.0759	**0.0620**	**−18.3** %

(relative L^2 on the held-out material.) The head wins on every fold and every metric; MSE falls 30.6 %.

Two results deserve more attention than the headline number.

T_{s} improves more than τ does. The integrated kinetic energy — the quantity that actually enters the total energy — improves from 2.55 → 1.13 %, 0.53 → 0.27 % and 0.90 → 0.58 %: roughly halved on every fold, a larger relative gain than the pointwise error. That is the expected direction, since τ_{vW} is supplied exactly and contributes no model error to the integral.

It trains *faster*. 148.6 s per fold against 180.8 s, and to a lower final training loss. The extra FFT for τ_{vW} is more than repaid by conditioning: the network fits only τ_{P}, whose dynamic range is far narrower than τ's. This is not an accuracy-versus-cost trade.

Caveat

The constraint did *not* bind. The unconstrained baseline violated the bound at zero points in all three folds, with margins of 0.83–1.05 eV/Å^3. On converged reference data the head's gain therefore comes from handing the model 31 % of the answer analytically, not from rescuing violations. The hard guarantee is insurance that costs nothing here and becomes load-bearing inside an SCF loop, where the functional sees non-converged densities far from the training distribution. These three structures cannot distinguish "the head enforces the bound" from "the bound is easy here"; the enforcement claim rests on the tests, which check it at random initialisation, at 50× weights and after training at a destructive learning rate.

7.3 Soft constraints

Where a constraint cannot be built in, it enters the objective:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda_{\text{EL}}\mathcal{L}_{\text{EL}} + \lambda_{\text{TF}}\mathcal{L}_{\text{TF}} + \lambda_T\mathcal{L}_T + \lambda_{\text{scale}}\mathcal{L}_{\text{scale}}. \tag{7.3}$$

Every physics weight defaults to zero, so the objective is exactly the supervised baseline until a term is enabled deliberately. Three design rules:

1. **Physics terms act on the prediction decoded to physical units**, never on the normalised representation — a constraint is a statement about physics, not about preprocessing.

2. **Every term is dimensionless.** A raw $(\text{eV}/\text{Å}^3)^2$ penalty varies by orders of magnitude between a light semiconductor and a transition-metal oxide, and no single weight could serve both.

3. **The data term is H^1, not L^2**, when gradients matter: τ_{vW} depends on $\nabla\rho$, and gradient noise is amplified by $|\nabla\rho|^2/8\rho$ exactly where ρ is small.

7.3.1 The Euler–Lagrange residual

The most valuable soft term is Eq. (2.2) itself. Since μ is a constant, subtracting the cell average eliminates it and leaves a residual that must vanish for the exact density:

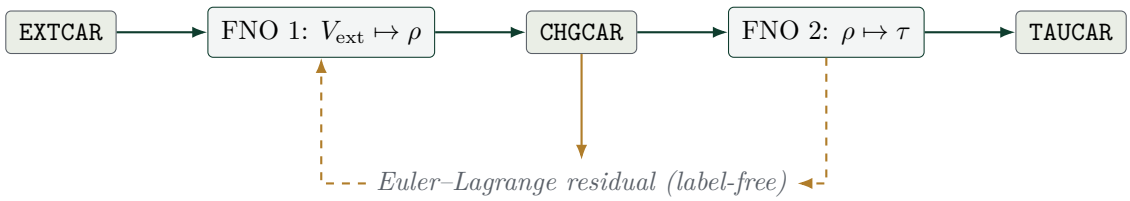
$$r(\mathbf{r}) = \frac{\delta T_{\text{s}}}{\delta\rho} + V_{\text{ext}} + v_{\text{H}}[\rho] + v_{\text{xc}}[\rho] - \overline{(\cdots)}. \tag{7.4}$$

It contains only V_{ext} (the network input) and ρ (its output), so it requires **no additional labels**.

> **Caveat**
> The residual is only as good as the kinetic functional used to build it. With the analytic $\text{TF} + \lambda\,\text{vW}$ surrogate it supplies a physically correct *inductive bias*, not ground truth, and must carry a modest weight.

7.4 The closed loop

The end state is not two models but one differentiable orbital-free cycle. The link is Eq. (2.2) with $\delta T_{\text{s}}/\delta\rho$ obtained by autograd through Model 2, since $T_{\text{s}} = \int \tau_{\theta_B}[\rho]\,\mathrm{d}\mathbf{r}$:



Three consequences, in order of practical value:

1. **Model 2 is regularised on Model 1's output distribution.** A functional trained only on converged reference densities fails when placed inside an SCF loop, because it then sees non-converged densities it was never shown. Joint training on the residual exposes it to exactly that distribution.

2. **The optimised quantity becomes the used quantity.** Orbital-free DFT needs $\delta T_s/\delta\rho$, not τ; training through the derivative removes the objective mismatch of Chapter 2.

3. **Inference becomes physically consistent.** A predicted (ρ, τ) pair satisfies the variational condition it is supposed to satisfy, rather than merely resembling the training data.

Beyond that, replacing Model 1's forward pass with a fixed-point solve of Eq. (2.2) using the learned functional — a deep-equilibrium model, differentiated by the implicit function theorem — would make the network an orbital-free *solver* rather than a regressor, with transferability bounded by the functional alone.

7.5 Why this is cheap here

Physics-informed training is normally expensive: derivatives must come from autograd through the network or from finite differences, and the latter injects a truncation error that the loss misattributes to the model. On a plane-wave grid neither problem arises.

Operator	On this grid	Cost
∇	$i\mathbf{G}$	one FFT, **exact**
∇^2	$-G^2$	one FFT, **exact**
$v_H[\rho]$	$4\pi e^2\rho(\mathbf{G})/G^2$	one FFT, **exact**
τ_{vW}	$\lvert\nabla\rho\rvert^2/8\rho$	one FFT, **exact**

These are exact for band-limited fields — precisely what a plane-wave grid carries — and the operator is already computing FFTs in every layer. The physics and the architecture share one mathematical substrate. That is the strongest technical argument for a Fourier neural operator here, independent of accuracy.

7.6 Roadmap

Phase	Work	Status
0	supervised baseline, honest split by material	done
1	hard constraints: log head, charge normalisation, Pauli head	Pauli head done
2	H^1 data loss; scaling-relation augmentation	available
3	Euler–Lagrange residual with the analytic surrogate	available
4	$\delta T_s/\delta\rho$ by autograd; joint training	planned
5	cycle consistency via density-to-potential inversion	planned
6	deep-equilibrium orbital-free solver	stretch

Phases 1–3 are low risk and independently valuable. Phase 4 is the research contribution.

CHAPTER 8

The role of Model 2: a learned kinetic energy functional

8.1 The question, stated without charity

A skeptic's version of the objection is worth writing down, because the weak answers to it are the ones most often given:

> *You already have Model 1, which predicts ρ from the geometry. If you have a converged DFT calculation you have τ for free; and if you do not have one, you have no ρ to feed Model 2 either. What is* `chg2tau` for*?*

Three answers are available. Two are weak.

"τ is a useful observable" — weak. It enters meta-GGA functionals, the electron localisation function and bonding analysis. True, but anyone holding a converged `CHGCAR` also holds a `TAUCAR`. Predicting a quantity one already has is a benchmark, not an application.

"It completes the pipeline" — weak. Chaining $V_{\text{ext}} \to \rho \to \tau$ yields τ for a structure never computed, which is useful for screening. But it makes Model 2 a convenience: nothing in it is more than interpolation, and its errors feed back into nothing.

"It *is* the orbital-free kinetic energy functional" — the actual reason. See below.

$T_{\text{s}}[\rho]$ is the *only* term of the total energy with no accurate explicit density functional. Kohn–Sham theory evades the problem by reintroducing orbitals and pays $\mathcal{O}(N^3)$ for it. Orbital-free DFT removes them and scales near-linearly, but is only as good as its functional. A model of $\rho \mapsto \tau$ **is** such a functional, because

$$T_{\text{s}}[\rho] = \int \tau[\rho](\mathbf{r})\,\mathrm{d}\mathbf{r}. \tag{8.1}$$

Model 2 is therefore not a predictor of a convenient observable; it is a candidate solution to the central open problem of orbital-free DFT, in the one form a network can represent and autograd can differentiate. **Model 2 is the scientific contribution; Model 1 is the infrastructure around it.**

8.2 Is τ a functional of ρ?

Worth settling, because the answer is subtler than "Hohenberg–Kohn says so".

For the integrated quantity the argument is clean: HK gives $\rho \to v_{\text{s}}$ uniquely for non-interacting v-representable densities, v_{s} determines the orbitals, and the orbitals determine T_{s}. The same

chain $\rho \to v_s \to \{\psi_i\} \to \tau$ establishes $\tau[\rho]$ as well. But note what that chain contains — solving the Kohn–Sham equations. The functional is thus

- **exact but non-local**: τ at one point depends on ρ everywhere, because the orbitals are delocalised Bloch states;
- **not semi-local**: no finite gradient expansion converges to it, which is precisely why Thomas–Fermi and its gradient corrections fail;
- **expensive to evaluate exactly**: the definition requires the very diagonalisation orbital-free DFT exists to avoid.

Two architectural consequences follow. The architecture *must* be non-local, which rules out a finite-receptive-field convolutional network and motivates the globally coupled Fourier layer of Chapter 3. And the target is well posed: unlike most machine-learning targets, $\tau[\rho]$ is a genuine single-valued functional, so failure to learn it is a capacity or data problem, not an ill-posedness problem.

> **Caveat**
> τ is a functional of ρ *for ground-state, non-interacting v-representable densities.* Inside a self-consistency loop the intermediate densities are not ground states of any v_s, so strictly $\tau[\rho]$ is undefined there. Every orbital-free calculation uses the functional anyway — but it means a model trained only on converged densities is *extrapolated* every time it is used variationally. This is the single largest risk in the programme.

8.3 What the model must get right — and it is not τ

Orbital-free DFT does not consume τ. It consumes the functional derivative

$$v_s^{\text{kin}}(\mathbf{r}) \equiv \frac{\delta T_s}{\delta \rho(\mathbf{r})}, \tag{8.2}$$

which is the term appearing in Eq. (2.2) and therefore the term that determines the density.

A model can have small $\|\tau_\theta - \tau\|$ and a *useless* derivative, because differentiation amplifies high-frequency error: a ripple of amplitude ϵ and wavevector G contributes ϵ to the value but ϵG to the gradient. Three consequences:

1. Reporting only pointwise τ error reports a **proxy**. The relative L^2 figures of Chapter 6 are necessary, not sufficient.
2. The H^1 data loss exists for this reason: it penalises the gradient error that plain L^2 ignores.
3. The definitive test is **downstream** — insert the functional into an orbital-free minimiser and measure the converged density and energy.

8.4 The mechanism: $\delta T_s/\delta\rho$ by autograd

This is what makes a *neural* functional more useful than a tabulated one, and it is mechanically simple: T_s is a scalar computed from ρ by a differentiable program, so one backward pass yields $\partial T_s/\partial \rho_i$ at every grid point — no finite differences and no derivative to derive by hand.

```
rho = rho_physical.detach().requires_grad_(True)      # (B,1,Nx,Ny,Nz)

tau = target_transform.inverse(model2(input_transform(rho), cell))
T_s = integrate(tau, cell)                            # (B,), eV

grad, = torch.autograd.grad(T_s.sum(), rho, create_graph=True)
dTs_drho = grad / volume_element                      # see below
```

8.4.1 The discretisation factor

Take $T_\mathrm{s} = \sum_i \tau_i \Delta v$. The functional derivative is defined by

$$\delta T_\mathrm{s} = \int \frac{\delta T_\mathrm{s}}{\delta \rho}\, \delta\rho \, \mathrm{d}\mathbf{r} \approx \sum_i \left(\frac{\delta T_\mathrm{s}}{\delta \rho}\right)_i \delta\rho_i \, \Delta v,$$

while autograd returns $\delta T_\mathrm{s} = \sum_i (\partial T_\mathrm{s}/\partial \rho_i)\, \delta\rho_i$. Hence

$$\boxed{\frac{\delta T_\mathrm{s}}{\delta \rho}(\mathbf{r}_i) = \frac{1}{\Delta v}\frac{\partial T_\mathrm{s}}{\partial \rho_i}} \qquad \Delta v = \frac{\Omega}{N_1 N_2 N_3}. \tag{8.3}$$

Caveat

Omitting the $1/\Delta v$ rescales the entire kinetic potential by the number of grid points — a factor of 3×10^4 on these grids — which would silently destroy any Euler–Lagrange residual built from it, without raising anything. `create_graph=True` is likewise required if the derivative is to appear in a loss that is itself backpropagated.

8.4.2 Validation of the derivative

The derivative is meaningful only if verified independently:

Test	Expectation
Substitute analytic Thomas–Fermi for Model 2	recovers $\frac{5}{3}C_\mathrm{TF}\rho^{2/3}$
Substitute von Weizsäcker	recovers $-\frac{1}{2}\nabla^2\sqrt{\rho}/\sqrt{\rho}$
Finite-difference a few random voxels	agrees to FD accuracy
Uniform density	$\delta T_\mathrm{s}/\delta\rho$ constant in space

These are cheap and they catch the Δv error, a broken transform chain, and any accidental `detach()`.

8.5 Coupling: how Model 2 trains Model 1

8.5.1 The equation

The Euler–Lagrange condition (2.2) holds pointwise at the ground state with μ constant in space. Every term is available:

Term	Source	Exact?
V_{ext}	Model 1's *input*	exact (tabulated pseudopotential)
ρ	Model 1's *output*	learned
$v_{\text{H}}[\rho]$	$4\pi e^2\rho(\mathbf{G})/G^2$, one FFT	**exact**
$\delta T_{\text{s}}/\delta\rho$	autograd through Model 2	learned
$v_{\text{xc}}[\rho]$	LDA/GGA, or omitted	approximate
μ	eliminated by the cell average	—

Subtracting the cell average removes the unknown μ exactly, leaving the residual of Eq. (7.4), which must vanish for the true ground-state density.

8.5.2 Why this is worth the trouble

The residual is computable from **Model 1's own input and output**, plus Model 2. It requires **no additional labels** — no extra DFT calculations, no targets — so it can be evaluated on unlabelled structures, on perturbed or interpolated geometries generated on the fly, and on the model's own predictions during training.

That converts Model 2 from a passive predictor into an **active physical constraint on Model 1**. This coupling is what the architecture is built around.

8.5.3 The loss

$$\mathcal{L}_1 = \underbrace{\left\|\rho_\theta - \rho^{\text{DFT}}\right\|_{\text{rel}}}_{\text{data}} + \lambda_{\text{EL}} \underbrace{\left\langle \left(r(\mathbf{r})/s\right)^2 \right\rangle}_{\text{Euler–Lagrange}} + \lambda_N \underbrace{\left(\frac{\int \rho_\theta - N}{N}\right)^2}_{\text{particle number}} \tag{8.4}$$

with $s = \text{std}(V_{\text{ext}})$ rendering the residual dimensionless and comparable across materials whose potentials differ by an order of magnitude. Three rules apply, each established elsewhere in this project: physics terms act on the prediction decoded to *physical* units; every term is dimensionless; and λ_{EL} is ramped only after the data term has converged, since a residual evaluated at random initialisation is noise.

8.5.4 Freeze Model 2, or train jointly?

Frozen. Model 2 is a fixed physics oracle. Simple, stable, and the constraint is exactly as good as the functional. Recommended first.

Joint. Mutual regularisation, and Model 2 is exposed to the densities Model 1 actually produces — which directly attacks the distribution-shift problem. But there is a real hazard.

> **Caveat**
>
> **The Euler–Lagrange residual alone has trivial solutions.** With both models free, the pair can drive $r \to 0$ by co-adapting: Model 2 can learn a functional whose derivative cancels $V_{\text{ext}} + v_{\text{H}} + v_{\text{xc}}$ for whatever ρ Model 1 emits, without either being correct. The residual is a *consistency* condition, not a *correctness* condition. Joint training is safe only with both data terms retained and dominant; the diagnostic of collapse is a residual that falls while the held-out data error rises.

8.6 Critical assessment

8.6.1 Genuine strengths

- The target is a well-posed functional, not a fitted correlation.
- It attacks the actual bottleneck of orbital-free DFT.
- The derivative is free and exact via autograd — removing the traditional obstacle to proposing new kinetic functionals, which is deriving $\delta T_s/\delta\rho$ by hand.
- Exact constraints exist and can be imposed structurally: $\tau \geq \tau_{vW}$ already is, supplying $\sim 31\,\%$ of τ analytically.
- On this data it beats every classical functional by roughly an order of magnitude, with Thomas–Fermi at negative R^2.

8.6.2 Real weaknesses

- **Distribution shift** is the central risk, and it is a domain question rather than a practical inconvenience: the functional is strictly undefined off the v-representable manifold.
- **We optimise τ and use $\delta T_s/\delta\rho$** (Section 8.3). Until training runs through the derivative, the objective is a proxy.
- **The derivative is unvalidated.** Nothing in the results of Chapter 6 bears on $\delta T_s/\delta\rho$; the Section 8.4 checks are not yet implemented.
- **Data scale.** Three structures of one element.
- **The Pauli potential constraint is unused.** Levy–Ou-Yang gives $v_P = \delta T_P/\delta\rho \geq 0$ — a constraint on the *derivative*, which is exactly the object that should be constrained.

8.7 Verification plan

Falsifiable, in dependency order:

1. **Derivative correctness** — the substitution tests above. Blocking: nothing downstream is meaningful until they pass.
2. **Residual floor** — evaluate $r(\mathbf{r})$ with the *reference* ρ and a classical functional. It will not vanish, since the functional is approximate; record the floor, because a learned functional must beat it to be adding anything.
3. **Ablation** — $\lambda_{EL} = 0$ versus > 0 on held-out material, matched otherwise. Same protocol as the Pauli-head comparison of Section 7.2.
4. **Label efficiency** — the real promise. Train Model 1 on k labelled structures plus m unlabelled ones under the residual, and show the curve beats k labelled alone. If the residual cannot buy label efficiency, its main practical argument fails.
5. **Downstream** — insert the functional into an orbital-free minimiser and measure converged density and energy error. The only test that matters.

8.8 Status

Component	State
$\rho \mapsto \tau$ supervised	done (rel. L^2 0.0620, R^2 0.9924)
$\tau \geq \tau_{\mathrm{vW}}$ structural	done (`PauliResidualOperator`)
Exact spectral ∇, ∇^2, v_{H}	done
Euler–Lagrange residual, analytic functional	done
$\delta T_{\mathrm{s}}/\delta\rho$ via autograd	**done** (`kinetic_potential`, `operator_kinetic_potential`)
Euler–Lagrange residual, *learned* functional	done in the library (`kinetic=`); not wired into training
Joint training	not implemented
Pauli potential $v_{\mathrm{P}} \geq 0$	not implemented
Orbital-free minimiser integration	not implemented

The autograd derivative is what turns two regressions into a differentiable orbital-free solver, and it is exact: against analytic Thomas–Fermi it reproduces the closed-form potential to 1×10^{-14} eV, and it satisfies Euler's homogeneity sum rule $\int (\delta T_{\mathrm{s}}/\delta\rho)\, \rho\, d^3r = \frac{5}{3} T_{\mathrm{s}}$ to ten decimal places. On a rough density it is *more* reliable than the hand-derived analytic von Weizsäcker potential, which drifts from that sum rule by several per cent where the autograd route stays exact.

Accuracy in τ does not imply accuracy in its derivative

Measured on a model trained to reproduce analytic Thomas–Fermi: relative L^2 of 3.0×10^{-3} on τ, and 9.2×10^{-1} on $\delta T_{\mathrm{s}}/\delta\rho$ — a 307× amplification, with the *mean* wrong by a factor of fifteen.

A mean that wrong is not high-frequency amplification, which would give noise about the right value. It is a direction the training data never explored: the derivative's mean is exactly the response of T_{s} to a *uniform* rescaling of ρ, and every training density had the same mean. Retrained on densities whose mean varies, the same architecture with the same τ error gave 1.6×10^{-1}, and the sum rule went from 0.057 to 0.973.

The sum rule is the diagnostic, and it is computable from a learned functional with no reference potential — the only check available when there is no closed form. Read it before closing the loop.

8.9 Distilling the Pauli factor, and which candidate to quote

What Model 2 has to get right is the Pauli enhancement factor $F = (\tau - \tau_{\mathrm{vW}})/\tau_{\mathrm{TF}}$, so that is what the symbolic search fits. A search does not return *an* expression: it returns a front, every candidate that was the best of its length. The front states a trade and refuses to resolve it.

The **knee** is where resolving it costs least — the candidate nearest the ideal corner $(0, 0)$ once both axes are rescaled to $[0, 1]$ across the front:

$$d_i = \sqrt{\hat{c}_i^2 + \hat{\mathcal{L}}_i^2}, \qquad \hat{x}_i = \frac{x_i - \min_j x_j}{\max_j x_j - \min_j x_j}.$$

Two choices in that definition are load-bearing.

Both axes are rescaled. A node count and a loss have no common unit, so a Euclidean distance between them is meaningless until they are made comparable. Rescaling across the front is what gives the metric a meaning relative to the range the search actually produced.

The loss enters logarithmically. A front spans orders of magnitude. On a linear scale every candidate but the most accurate collapses onto $\hat{\mathcal{L}} \approx 1$, and the knee degenerates to the shortest expression whatever it cost — which is not a trade-off, it is a constant.

The report carries both candidates: the lowest-loss expression and the knee, each with its own parity plot on identical voxels, laid side by side. Where the two clouds are indistinguishable the extra nodes bought nothing, and that is visible rather than summarised.

> **Note**
> A knee is a heuristic, not a theorem. With one candidate it is that candidate; where the front is a straight line every point is equidistant. It answers *which expression is worth its length.* Whether an expression is *right* is what the asymptotic limits test — and those are statements about physics the search was never shown.

CHAPTER 9

The training pipeline

9.1 Requirements

> **Note**
> Poraquê requires **Python 3.11 or newer** (`requires-python = ">=3.11"`). Every module has been verified against the 3.11 grammar, so nothing in the source uses syntax or standard-library calls newer than that. There is deliberately no upper bound: newer interpreters are supported.

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

This pulls in NumPy, SciPy, ASE, Matplotlib, PyYAML and PyTorch, together with `mp-api` and `python-dotenv` for the Materials Project ingestion of Chapter 5. A working `pdflatex` or `latexmk` is needed only for the automatic PDF reports of Section 9.7; without one the pipeline still runs and simply writes the LaTeX source instead.

Four console commands are registered: `poraque-train`, `poraque-inference`, `poraque-committee` and `poraque-mp`. The last fetches charge densities for a chemical space; its `-estimate` flag is a pure dry run that reports the exact transfer size on the console and writes nothing, and its output directory defaults to the current one.

9.2 Configuration

A run is defined by a YAML file with four sections — `data`, `model`, `training`, `output` — mirroring the four things a run needs. Command-line flags override individual entries, so one committed configuration can be swept from the shell without being edited or copied:

```
poraque-train --config configs/train.yaml
poraque-train --config configs/train.yaml --epochs 500
```

Unknown keys are *rejected* rather than 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.

9.3 The external potential is computed, not imported

Poraquê **computes** the external potential natively, from `POSCAR`, `INCAR` and `POTCAR`, using the tabulated pseudopotential of Chapter 5. There is no option to import one: the training input must be exactly what the pipeline produces at inference time, when no such file exists.

This costs essentially nothing in fidelity. Against reference potentials on five platinum structures:

Structure	grid	relative L^2	Pearson r
struct_000	128^3	2.13×10^{-5}	1.000000
struct_001	$120\times128\times128$	5.95×10^{-5}	1.000000
struct_002	$128\times120\times128$	5.75×10^{-5}	1.000000
struct_003	128^3	5.66×10^{-5}	1.000000
struct_004	128^3	5.36×10^{-5}	1.000000

Implementation

The shared grid is taken from `CHGCAR`, not `EXTCAR`: the density is present in any standard run, the external potential is not. Sourcing the grid from a file that may not exist would have reintroduced the dependency through the back door.

9.3.1 Data that ships no pseudopotentials

The construction above needs a `POTCAR`, and a public density archive does not have one. `data.potcar_dir` names a `POTCAR` *library* to stand in — one entry per species, as VASP distributes them — so a Materials Project download gets the same tabulated potential a local run does, from the structure the density carries in its own header. The library is consulted only where the data itself has nothing; a run with its own `POTCAR` ignores it.

Without a library the Gaussian pseudo-ion model is used instead, at a relative L^2 of order 10^{-1} rather than 10^{-5}, and the two constructions differ from each other by 0.38 on the Ag–Pt–Pt set. Section 5.4 works through what that does and does not license; the short version is that it remains a well-posed learning problem but stops being the Hohenberg–Kohn map of VASP's V_{ext}.

Because these are different physical quantities rather than different roundings of one, they must never share a cache: the cache directory name records which was used (`res32_potcar`), and a dataset mixing sources that disagree raises a warning naming both.

9.4 Gaussian smoothing of the potential

The tabulated pseudopotential is sharply peaked at the ionic cores. Since a network with a fixed mode truncation may not use the highest spatial frequencies, an optional Gaussian blur is available:

```
poraque-train --config configs/train.yaml \
       --gaussian-blur 0.15 --blur-method spectral
```

Smoothing is applied *on top of* the tabulated potential; it never replaces it. It is off by default (`gaussian_blur: null`).

9.4.1 Two methods, and why the default matters

`spectral` (default) multiplies by $e^{-G^2\sigma^2/2}$ in reciprocal space. Exact for a band-limited field, and it uses the true reciprocal metric, so the blur stays isotropic in Cartesian space on *any* cell.

`ndimage` uses `scipy.ndimage.gaussian_filter` with `mode="wrap"` — the wrap is essential, since the cell is a torus.

> **Caveat**
> The two agree to a relative 1.4×10^{-5} on an *orthogonal* cell and differ by 2–6 % on the fcc cell of the reference data. `ndimage` blurs along *lattice* axes, which is anisotropic in Cartesian space whenever the lattice vectors are not orthogonal. For a face-centred cubic lattice — angles of 60° — it is measurably wrong, not merely different.

9.4.2 Measured effect

The blur is far from uniform. Binned by distance to the nearest ion at $\sigma = 0.15$ Å:

Shell [Å]	voxels	mean $\lvert\Delta V\rvert$ [eV]	$\overline{V}$ [eV]	$\overline{\rho}$ [e/Å^3]
0–0.5	959	9.64	−113.8	3.51
0.5–1.0	6748	4.06	−45.8	1.46
1.0–1.5	17854	1.58	+13.0	0.32
1.5–2.0	7194	1.18	+25.8	0.16

The core region (< 1 Å) is 23.5 % of the voxels but absorbs 50 % of the total change, and the deepest well shrinks by 7.8 % ($-137.3 \to -126.6$ eV). Those same voxels carry the charge-density peaks ($\overline{\rho} = 1.72$ against a cell mean of 0.61). Smoothing therefore preferentially removes exactly the information that generates the density maxima.

> **Note**
> On a matched held-out comparison — same four training structures, same seed, architecture and epochs, sole occupancy of the accelerator — blurring changed the held-out relative L^2 from 0.0274 to 0.0272, i.e. by 0.7 %, and training time by 0.4 %. **Neither is a meaningful difference.** At 32^3 the field is already band-limited at $|G|_{\max} \approx 26$ Å^{-1} and the blur removes content the mode truncation was discarding anyway. The option is worth revisiting only at higher resolution or larger `modes`, where the model could actually use those frequencies.

9.5 Training

One model per task is trained across all the training structures, never one per material.

```
poraque-train --config configs/train.yaml
```

There is one protocol and one variation on it: `valid_fraction` holds back a fifth of the structures by default, and `enable_kfold` (Section 9.6) swaps the whole thing for cross-validation. Nothing else selects a training mode.

Batches are drawn *across* materials: the sampler groups structures by grid shape and shuffles both within and across those groups, so a gradient step generally mixes several. On the reference

data the buckets are $(32, 32, 32) \times 3$, $(30, 32, 32) \times 1$ and $(32, 30, 32) \times 1$ — three distinct shapes flowing through one set of weights.

The run writes a single unified checkpoint, `models/poraque_models.poraque`, holding both operators under the keys `ext2chg` and `chg2tau`.

> **Caveat**
> With `valid_fraction: 0` every structure is in training, so the reported metrics are **training fit**. They show the model can represent the data; they are not a generalisation estimate. Raise `valid_fraction`, or use the cross-validation of Section 9.6, for that. Measured on seventeen platinum structures: the all-structure training fit gives relative L^2 of 0.0209 (`ext2chg`) and 0.0140 (`chg2tau`), against held-out values of 0.0379 and 0.0511 — roughly two to four times larger. That gap is the memorisation margin, and quoting the training-fit number as a result would be a serious overstatement.

9.6 K-fold cross-validation

```
poraque-train --config configs/train.yaml \
       --kfold --k-folds 5
```

or in the configuration file:

```
training:
  enable_kfold: true
  k_folds: 5
```

Each fold trains a fresh model on $K - 1$ groups and scores it on the held-out group, in physical units. The result is a generalisation estimate *with a spread*, not a model to deploy.

9.6.1 The split is at the structure level

This is the part that decides whether the number means anything.

A fold is a set of *material names*; every voxel of a material goes to the same side of the split. Splitting at the voxel level would place the same material in both training and validation, and since neighbouring voxels of one crystal are enormously correlated, the model would effectively be scored on data it had already seen. The resulting figure would look excellent and would say nothing about transfer to a new material.

> **Implementation**
> `k_folds` is capped at the number of structures; setting it equal to that count *is* leave-one-out. The partition is verified exact and disjoint — every structure appears in exactly one validation fold.

9.7 Automatic PDF reporting

Every run assembles its metrics and figures into a typeset PDF under `models/<name>/report/`. In cross-validation mode a single *consolidated* report covers all folds, with the per-fold table and

the mean ± standard deviation of each metric.

The mechanism matters for hygiene: the LaTeX source is written into a `tempfile.mkdtemp()` scratch directory, compiled there, and only the PDF is moved out; the whole tree is then removed with `shutil.rmtree`.

> **Implementation**
> Deleting the directory wholesale is the only reliable way to guarantee that no `.tex`, `.aux`, `.log`, `.toc`, `.fls` or `.fdb_latexmk` survives — a hand-maintained list of extensions would drift as packages begin emitting new ones. Compilation falls back from `latexmk` to two `pdflatex` passes, and if no toolchain is present the source is written beside the target and the caller is told, rather than failing silently.

Each report carries a *What these numbers do not show* section, generated from the run itself: whether anything was held out, how many structures there were, and whether the figures support a generalisation claim.

9.8 Outputs

Path	Contents
`models/poraque_models.poraque`	both trained operators, in one file
`models/<name>/<name>.poraque`	the trained operators, in one file
`models/<name>/log/*.log`	human-readable training log
`models/<name>/log/*.json`	machine-readable metrics and full history
`models/<name>/log/*_config.yaml`	the resolved configuration, for reproduction
`models/<name>/plots/`	loss curves, cross-sections, parity plots
`models/<name>/report/`	typeset report per model or per cross-validation

CHAPTER 10

From fields to energies

Everything up to this point produces *fields*. This chapter closes the loop: it assembles ρ, τ and V_{ext} into the Kohn–Sham total-energy expression, states precisely which terms Poraquê can and cannot evaluate, and quantifies how accurate the fields must be before the resulting energy means anything. The implementation is `poraque.physics.energy`; the ASE interface built on it is `poraque.calculator.Poraque`.

10.1 The expression

$$E = \underbrace{\int \tau \, d^3r}_{T_{\text{s}}} + \underbrace{\int \rho V_{\text{ext}} \, d^3r + E_{\alpha Z}}_{E_{\text{ext}}} + \underbrace{\frac{1}{2}\int \rho v_{\text{H}} \, d^3r}_{E_{\text{H}}} + E_{\text{xc}}[\rho] + E_{\text{Ewald}} \tag{10.1}$$

Only the first term uses τ, and it uses it in the simplest possible way: T_{s} *is* the integral of the kinetic energy density. That is the whole reason `chg2tau` is worth training — it makes the one term of the Kohn–Sham functional that has no closed form available from the density alone.

10.1.1 Hartree

Poisson in reciprocal space,

$$v_{\text{H}}(\mathbf{G}) = \frac{4\pi e^2 \rho(\mathbf{G})}{G^2}, \qquad v_{\text{H}}(\mathbf{G} = 0) = 0, \tag{10.2}$$

so $E_{\text{H}} = \frac{1}{2}\int \rho v_{\text{H}} \, d^3r$. Dropping $\mathbf{G} = 0$ is the neutralizing-background convention, chosen to match `ExternalPotential` so the two potentials are directly addable.

> **Implementation**
> The implementation is validated against an exact case rather than a recorded output: for $\rho = \rho_0 + A\cos(\mathbf{G}\cdot\mathbf{r})$ the energy is $\pi e^2 A^2 \Omega / G^2$ in closed form, and the FFT result reproduces it to 8×10^{-16} relative. A uniform density gives exactly zero, which is the correct answer in this convention rather than a numerical accident.

10.1.2 Exchange and correlation

Two rungs of the ladder are implemented, selected by the `functional` argument.

LDA. Dirac exchange plus Perdew–Wang 1992 correlation, both local:

$$\varepsilon_{\text{x}} = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{1/3} \rho^{1/3}, \qquad \varepsilon_{\text{c}}(r_s) = -2A(1 + \alpha_1 r_s)\ln\left[1 + \frac{1}{2A\,\Phi(r_s)}\right], \tag{10.3}$$

with $\Phi(r_s) = \beta_1 r_s^{1/2} + \beta_2 r_s + \beta_3 r_s^{3/2} + \beta_4 r_s^2$ and $r_s = (3/4\pi\rho)^{1/3}$.

PBE (the default). A generalized-gradient correction built on the same local pieces:

$$E_{\mathrm{x}}^{\mathrm{PBE}} = \int \rho\, \varepsilon_{\mathrm{x}}(\rho) F_{\mathrm{x}}(s)\, d^3r, \qquad F_{\mathrm{x}}(s) = 1 + \kappa - \frac{\kappa}{1 + \mu s^2/\kappa}, \tag{10.4}$$

$$E_{\mathrm{c}}^{\mathrm{PBE}} = \int \rho\, [\varepsilon_{\mathrm{c}}(r_s) + H]\, d^3r, \qquad H = \gamma \ln\left[1 + \frac{\beta}{\gamma} t^2 \frac{1 + At^2}{1 + At^2 + A^2 t^4}\right], \tag{10.5}$$

with $A = (\beta/\gamma)\left[e^{-\varepsilon_{\mathrm{c}}/\gamma} - 1\right]^{-1}$, the reduced gradients $s = |\nabla\rho|/(2k_{\mathrm{F}}\rho)$ and $t = |\nabla\rho|/(2k_s\rho)$, $k_{\mathrm{F}} = (3\pi^2\rho)^{1/3}$ and $k_s = \sqrt{4k_{\mathrm{F}}/\pi}$. The spin-scaling factor ϕ is 1 throughout: Poraquê's fields are unpolarized. PBE has no fitted parameters — κ is fixed by the Lieb–Oxford bound, $\mu = \beta\pi^2/3$ and $\gamma = (1 - \ln 2)/\pi^2$ by the uniform gas.

Implementation

PBE is the default because it matches the data: the reference calculations use `PAW_PBE` pseudopotentials with `LEXCH = PE`, so ρ and τ are PBE quantities. An LDA E_{xc} on a PBE density is neither a PBE energy nor an LDA one. On the reference supercells the two differ by -0.92 eV/atom, 0.65 % of E_{xc}.

The defining limit is also the sharpest test: as $\nabla\rho \to 0$, $F_{\mathrm{x}} \to 1$ and $H \to 0$, so PBE collapses onto LDA *bit-exactly* on a uniform density. The implementation reproduces that, along with $\varepsilon_{\mathrm{x}} = -0.458165293/r_s$ Ha per electron exactly, PW92's published table to 10^{-6}, and the analytic s for a cosine density to 10^{-15}.

Caveat

The semilocal terms need $\nabla\rho$ of a *predicted* field. That gradient carries the network's noise, and F_{x} amplifies it; on a band-limited grid it also aliases wherever the density has sharp core peaks. On the current models — Section 10.4 — the density is not accurate enough for its gradient to be meaningful, so the PBE-LDA difference should not yet be read as physics.

The density is clipped at zero in the algebra, but the gradient is taken of the *unclipped* field: clipping first would put a kink at every point where a band-limited density rings below zero, and spectral differentiation of a kink rings far worse than the undershoot it was meant to remove. $\rho^{4/3}$ of a negative number is not real, and the physical content of those points is "no electrons here".

10.2 The G = 0 bookkeeping

This is the subtle part of Equation 10.1, and getting it wrong produces a plausible number that is wrong by eV per atom.

Each of the three electrostatic terms — electron-ion, Hartree, ion-ion — diverges at $\mathbf{G} = 0$, and in a neutral cell the three divergences cancel exactly. The standard plane-wave treatment drops $\mathbf{G} = 0$ from all three and adds back the finite remainder of the electron-ion term. Writing VASP's local potential as

$$V_{\mathrm{ext}}(\mathbf{G}) = \frac{1}{\Omega} \sum_s S_s(\mathbf{G}) \left[v_{\mathrm{short}}^s(G) - \frac{4\pi Z_s^{\mathrm{val}} e^2}{G^2}\right], \tag{10.6}$$

the divergent piece is the second bracketed term and the finite remainder at $\mathbf{G} = 0$ is $S_s(0) = N_s$

times $v^s_{\text{short}}(0)$. Hence

$$E_{\alpha Z} = \frac{N_{\text{elec}}}{\Omega} \sum_s N_s \, \text{PSCORE}_s, \qquad \text{PSCORE}_s = \lim_{q\to 0} v^s_{\text{short}}(q), \tag{10.7}$$

which is exactly VASP's `alpha Z`. It is read straight from the `POTCAR` tables — the same tables Chapter 5 uses for the potential itself — via `PotcarSingle.pscore`.

> **Note**
> When the tables are unavailable the term is reported as `None`, and `EnergyComponents.missing` names it. Defaulting to zero would look like a working calculation while silently dropping a term of order eV per atom, which is precisely the size of the effects one would want to resolve.

10.3 Ewald

The ion-ion energy of point charges q_i in a neutralizing background:

$$\begin{aligned} E_{\text{Ewald}} = {} & \frac{e^2}{2} \sum_{i,j,\mathbf{R}}{}' \frac{q_i q_j \operatorname{erfc}(\eta|\mathbf{r}_{ij}+\mathbf{R}|)}{|\mathbf{r}_{ij}+\mathbf{R}|} + \frac{2\pi e^2}{\Omega} \sum_{\mathbf{G}\neq 0} \frac{e^{-G^2/4\eta^2}}{G^2} |S(\mathbf{G})|^2 \\ & - \frac{\eta e^2}{\sqrt{\pi}} \sum_i q_i^2 - \frac{\pi e^2}{2\eta^2\Omega} \Big(\sum_i q_i\Big)^2. \end{aligned} \tag{10.8}$$

The last term is the interaction with the compensating background. It is not optional: without it the result depends on the arbitrary splitting parameter η instead of being invariant under it — which is also the cheapest available test that the implementation is correct.

> **Implementation**
> Verified against tabulated Madelung constants: rocksalt 1.747564595 to 8×10^{-13}, caesium chloride 1.762674773 to 7×10^{-11}, zincblende 1.6380550 to 5×10^{-8}; and against the jellium limit $E = -\xi q^2 e^2/2L$, $\xi = 2.8372974795$, for a single ion in a cube. Each depends on every part of the sum being individually right.

10.4 What the number means

10.4.1 It is not a DFT total energy

`CHGCAR` holds the valence *pseudo* density and `TAUCAR` the valence pseudo kinetic energy density. The PAW one-centre terms — the difference between the pseudo and all-electron descriptions inside the augmentation spheres — never appear in any file Poraquê reads, so they are absent from Equation 10.1 entirely. For the 27-atom Pt supercell the result is ≈ -31215 eV against a VASP `TOTEN` of order -100 eV. No adjustment of the terms above closes that gap; the missing physics is simply not in the inputs.

10.4.2 The cancellation problem

The individual terms are large and of alternating sign:

Term	eV
T_s	+6 207
E_{ext}	−12 635
$E_{\alpha Z}$	+1 753
E_H	+3 230
E_{xc}	−3 821
E_{Ewald}	−25 949
Total	−31 215

Across the reference supercells the *physically relevant* variation of that total is 0.27 eV/atom — a relative $\sim 10^{-4}$ of its own magnitude. That sets a brutal accuracy requirement on the fields, because every electrostatic term is linear or quadratic in ρ.

Measured on the earlier twelve-structure dataset, with the models evaluated on their own training data (so this is a lower bound on the true error). These figures have *not* been re-measured on the current seventeen-structure dataset:

Quantity	Value
True spread of E across the twelve structures	0.27 eV/atom
MAE of the predicted energy differences	0.29 eV/atom
Ratio error / signal	1.06
Correlation of predicted vs. true differences	$r \approx -0.1$

Caveat

The error is **comparable to the signal**, and the correlation is consistent with zero. Energy differences from this pipeline are not yet usable: the predicted ordering of structures carries no information. This is an improvement on the earlier 3× ratio measured on a smaller dataset, but the qualitative verdict is unchanged.

The cause is cancellation, not a defect in the energy module: a field-level relative L^2 of 2×10^{-2} cannot survive a 10^{-4} signal-to-total ratio. Reaching 0.01 eV/atom requires densities accurate to roughly 10^{-5} relative — three orders of magnitude beyond the current models.

Two candidate explanations were tested and rejected:

Grid resolution. On reference fields the total energy shifts by 0.08 eV out of 31 215 (0.003 eV/atom) between `resolution: 32` and the native 128^3 mesh, and both T_s and the electron count are preserved *exactly* — spectral resampling is an exact band-limited projection. The grid is not the limiting factor.

Electron-count drift. Predicted densities miss the exact valence count by up to 2.8 %. Rescaling ρ to the known total makes the result *worse* (3.20 eV/atom, from 0.83), because E_{ext} is linear in ρ and a 0.3 % rescale shifts a −12 690 eV term by 38 eV. The drift is a symptom, not the cause.

10.4.3 Consequences for the programme

This is the sharpest quantitative target the project has produced so far, and it reframes what "accurate enough" means. A relative L^2 of 0.03 on the density is an excellent *field* prediction and a useless *energy* one. Two directions follow, and they are not equivalent:

1. **Make the fields three orders of magnitude better.** Direct, and almost certainly

unreachable by scaling the current architecture.

2. **Make the energy insensitive to the field error.** Predict the energy *difference* from a reference state rather than the absolute total, or exploit the variational property of the Kohn–Sham functional: at the true density the energy is stationary, so a first-order density error produces only a second-order energy error. That property is available only if the energy is evaluated with a *self-consistent* functional, which is what the physics-informed programme of Chapter 7 is ultimately for.

The second route is why $\delta T_s/\delta\rho$ (Chapter 8) is worth having: it is the ingredient that turns a learned τ into a variational object rather than a fitted one.

10.5 The ASE interface

`poraque.calculator.Poraque` attaches the whole chain to an `ase.Atoms` object:

```
from ase.build import bulk
from poraque.calculator import Poraque

atoms = bulk("Pt", "fcc", a=3.92, cubic=True)
atoms.calc = Poraque("models/poraque_models.poraque",
                     potcar_dir="POTCARs")
energy = atoms.get_potential_energy()
```

Each call runs `Atoms` $\to$ `Structure` $\to$ `FieldGrid` $\to V_{\text{ext}} \to \rho \to \tau \to E$, and leaves the three fields on the calculator so a prediction can be written out rather than only reduced to a number.

It resembles MACE or NequIP at the interface but not underneath. `get_forces()` and `get_stress()` raise `NotImplementedError`, so geometry optimisation and molecular dynamics are out of reach. Two independent pieces are missing: $\partial V_{\text{ext}}/\partial \mathbf{R}_a$ (analytic, a Hellmann–Feynman term) and back-propagation of $\partial E/\partial\rho$ through both operators. A finite-difference stand-in is no shortcut — on a fixed grid it would be dominated by the grid's own discontinuity as atoms cross voxel boundaries.

Caveat

Without a `POTCAR` the calculator falls back to the Gaussian pseudo-ion model of Chapter 5, whose relative L^2 against the tabulated potential is 0.13 — four orders of magnitude worse than the 2×10^{-5} the operators were trained on. The input is then outside the training distribution and the prediction is not meaningful. The calculator warns once and proceeds, because the fallback remains useful for smoke tests.