

**Accurate electronic and chemical properties of 3d transition
metal oxides using a calculated linear response U and a
DFT+U(V) method: Supporting information**

Zhongnan Xu and John R. Kitchin^{*}

*Department of Chemical Engineering, Carnegie Mellon University,
5000 Forbes Ave, Pittsburgh, PA 15213*

Yogesh V. Joshi and Sumathy Raman

*Exxon-Mobil Research and Engineering,
1545 Route 22 E Ste 1, Annandale, NJ 08801*

(Dated: February 3, 2016)

CONTENTS

I. Introduction	3
II. Calculated linear response U values	4
III. $E_U(U)$ on atoms with QE and GBRV PP	6
IV. Band gaps with linear response U	9
V. DFT+ $U(V)$ for bulk cobalt monoxide	11
VI. Tables data needed for DFT+ $U(V)$ energy	18
VII. DFT+ $U(V)$ Oxidation energies of oxides	20
A. Fitting the x parameter to experimental data	22
B. Store test/training set oxidation energies in table	27
C. Store DFT/DFT+ $U(V)$ with confidence intervals in table	32
D. Graph test/training set and DFT/DFT+ $U(V)$ parity plot	37
VIII. Oxidation energies computed with DFT+ U and HSE06	40
A. DFT+ U total energies with element specific U values	40
B. HSE06 offset formation energies	41
C. Storing DFT+ U and HSE06 energies with respect to metal atom (eV/M) in a single table	42
D. Calculate oxidation reaction energies	44
IX. DFT+ $U(V)$ Formation energies of oxides	46
A. QE library PPs	46
B. GBRV PPs	53
X. Finding equilibrium bulk structures	61
A. Vanadium oxides	62
1. VO	62
2. V_2O_3	63
3. VO_2	64

4. V_2O_5	65
B. Manganese oxides	67
1. MnO	67
2. Mn_3O_4	68
3. MnO_2	70
C. Iron oxides	71
1. FeO	71
2. Fe_3O_4	74
3. Fe_2O_3	75
D. Nickel oxides	77
1. NiO	77
E. Cobalt oxides	79
1. CoO	79
2. Co_3O_4	79
XI. Behavior with respect to U for all systems	81
A. Original QE PP library	82
1. Atoms	82
2. Metals	88
3. Oxides	89
B. GBRV PPs	103
1. Atoms	103
2. Metals	109
3. Oxides	112
XII. Required modules	122
A. <code>espresso</code>	123
B. <code>ase_addons</code>	123
C. <code>pycse.py</code>	123
References	123


I. INTRODUCTION

In this supporting information, we have compiled all of the data and code required to make Figures 2, 4, and 5. We have also included sample python scripts that we used to perform total energy and linear response calculations. All of the required modules used in this article are found Section XII.

This supporting information also includes DFT+ $U(V)$ formation energies. We chose to not include this set of data in the final manuscript for two reasons. The improvement given by the DFT+ $U(V)$ approach for formation energies was not as universal as for oxidation energies. Furthermore, it is still unclear to us which metal materials require a linear response U approach, with there being a dearth of literature on this subject.

The organization is as follows. Section II contains a sample linear response U calculation along with the table with all the linear response U values we calculated in this study. Section III details the construction of Figure 2 of the manuscript, which demonstrates the pseudopotential dependent $E_U(U)$ behavior. Section V contains the data and code needed to generate the sample calculation of DFT+ $U(V)$ curve of CoO and generate Figure 4 from the manuscript. Section VI and VII contains the raw data along with the code to generate Figure 5 of the manuscript, which is the performance of the DFT+ $U(V)$ approach for predicting accurate oxidation energies between oxides. Finally, Section IX contains the application of the DFT+ $U(V)$ approach to formation energies of a large number of oxides.

Sections X and XI are large sections that contain sample code of finding the optimal geometries of some structures along with all of the raw data of the total energies at different U values. Section XII contains the modules one needs to run the python code contained in this supporting information.

The file that generated this supporting information can be found here: 

II. CALCULATED LINEAR RESPONSE U VALUES

We first present sample code for calculating the linear response U using the `espresso` module we have constructed (Section XII). The program used to compute the U was the `r.x` function written by Matteo Cococcioni and can be found at http://media.quantum-espresso.org/santa_barbara_2009_07/slides-exercices/W1d4_ksda_ldau.

[tar.gz](#). All bulk structures used for the calculation of the linear response U were the ground state structures calculated by DFT.

```

1  from jasp import *
2  from ase_addons.bulk import corundum
3  from espresso import *
4
5  bulk = corundum(('V', 'O'), vol=99.274, mags=(0.5, 0))
6
7  hubbard_Us = 1e-20 * np.ones(len(bulk))
8
9  with Espresso('linear-response/V203', atoms=bulk,
10               # CONTROL
11               calculation='scf',
12               restart_mode='from_scratch',
13               # SYSTEM
14               ecutwfc=60.0,
15               ecutrho=600.0,
16               occupations='smearing',
17               smearing='mp',
18               degauss=0.01,
19               lda_plus_u=True,
20               U_projection_type='atomic',
21               nspin=2,
22               Hubbard_U=hubbard_Us,
23               # ELECTRONS
24               mixing_beta=0.3,
25               conv_thr=1e-8,
26               kpts=(6, 6, 6)) as calc:
27      print calc.get_linear_response_Us(({0:(0, 1, 2, 3), 4:(4, 5, 6, 7, 8, 9)}))

```

The table below lists all of the linear response U calculations that we have calculated in this study. Additional materials than the ones presented in the manuscript we're studied in order to determine whether our method was suitable for formation energies. The DFT+ U (V) computed formation energies with varying x values is presented in Section [IX](#).

TABLE I. Calculated linear response U values of oxides. All linear response values are in eV.

System	$U_{\text{QE Library}}$	U_{GBRV}
VO	4.12	3.61
V ₂ O ₃	4.99	4.64
VO ₂	5.14	5.02
V ₂ O ₅	5.12	4.67
Cr ₂ O ₃	2.73	4.86
CrO ₃	4.99	7.42
MnO	4.94	5.52
Mn ₃ O ₄	4.05	6.11
MnO ₂	4.78	4.20
FeO	4.10	5.80
Fe ₃ O ₄	3.72	6.07
Fe ₂ O ₃	3.47	5.21
CoO	4.89	5.86
Co ₃ O ₄	5.43	7.44

TABLE II. Calculated linear response U values of metals. All linear response values are in eV.

System	$U_{\text{QE Library}}$	U_{GBRV}
V	3.73	3.59
Mn	N/A	5.94
Fe	2.64	5.22
Co	3.87	5.26
Ni	6.99	7.69

III. $E_U(U)$ ON ATOMS WITH QE AND GBRV PP

The code below extracts data out of tables in this file and relates the energetic contribution to the total energy with different values of U using tow sets of pseudopotentials. The data needed is the $E_U(U)$ data of isolated atoms of V, Mn, Co, and Fe using both the

QUANTUM-ESPRESSO original library of pseudopotentials [1] and GBRV high-throughput pseudopotentials [2]. The Hubbard U energy,

$$E_U = \frac{U}{2} \sum_{I,\sigma} \text{Tr}[\mathbf{n}^{I\sigma}(1 - \mathbf{n}^{I\sigma})], \quad (1)$$

is defined both by the U value as well as the occupations $\mathbf{n}^{I\sigma}$ of the electrons for given atom I with spin σ . These occupations are defined by the projector operator \mathbf{P}^I , and in QUANTUM-ESPRESSO, they are given as the atomic pseudo-wave-function,

$$\mathbf{P}^I = |\phi_m^I\rangle\langle\phi_{m'}^I|, \quad (2)$$

where $|\phi_m^I\rangle$ is the valence atomic orbital with angular momentum component $|lm\rangle$ of the atom sitting at site I and is determined by the specific pseudopotential used.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib import rc, rcParams
4
5  rc('xtick', labels=10)
6  rc('ytick', labels=10)
7
8  fig = plt.figure(1, (3.25, 3.25))
9  axes = fig.add_axes([0.19, 0.55, 0.3, 0.4])
10
11  U_V_atom_QE, E_V_atom_QE = np.array(zip(*V_atom_QE))
12
13  U_V_atom_GBRV, E_V_atom_GBRV = np.array(zip(*V_atom_GBRV))
14
15  E_V_atom_QE -= E_V_atom_QE[0]
16  E_V_atom_GBRV -= E_V_atom_GBRV[0]
17
18  axes.plot(U_V_atom_QE, E_V_atom_QE, marker='o', ls='none', c='gray', label='V QE PPs')
19  axes.plot(U_V_atom_GBRV, E_V_atom_GBRV, marker='s', ls='none', c='k', label='V GBRV PPs')
20  axes.set_ylabel(r'$E_{\{U\}}$ (eV)', size=10)
21  axes.set_xlim(-1, 7)
22  axes.set_xticklabels([])
23  axes.set_ylim(-0.2, 2.2)
24
25  axes.text(1, 1.5, 'V')
26
27  axes = fig.add_axes([0.51, 0.55, 0.3, 0.4])
28  axes.set_xticklabels([])

```

```

29 axes.set_ylim(-0.02, 0.12)
30 axes.set_yticks([0, 0.02, 0.04, 0.06, 0.08, 0.1])
31 axes.set_yticklabels([])
32
33 U_Mn_atom_QE, E_Mn_atom_QE = np.array(zip(*Mn_atom_QE))
34
35 U_Mn_atom_GBRV, E_Mn_atom_GBRV = np.array(zip(*Mn_atom_GBRV))
36
37 E_Mn_atom_QE -= E_Mn_atom_QE[0]
38 E_Mn_atom_GBRV -= E_Mn_atom_GBRV[0]
39
40 axes.plot(U_Mn_atom_QE, E_Mn_atom_QE, marker='o', ls='none', c='gray')
41 axes.plot(U_Mn_atom_GBRV, E_Mn_atom_GBRV, marker='s', ls='none', c='k')
42 axes.set_xlim(-1, 7)
43
44 axes.text(1, 0.08, 'Mn')
45
46 axes = axes.twinx()
47 axes.set_ylim(-0.02, 0.12)
48 axes.set_ylabel(r'$E_{U}$ (eV)', size=10)
49 axes.set_yticks([0, 0.02, 0.04, 0.06, 0.08, 0.1])
50
51 axes = fig.add_axes([0.19, 0.15, 0.3, 0.4])
52 U_Fe_atom_QE, E_Fe_atom_QE = np.array(zip(*Fe_atom_QE))
53
54 U_Fe_atom_GBRV, E_Fe_atom_GBRV = np.array(zip(*Fe_atom_GBRV))
55
56 E_Fe_atom_QE -= E_Fe_atom_QE[0]
57 E_Fe_atom_GBRV -= E_Fe_atom_GBRV[0]
58
59 axes.set_ylim(-0.3, 2.8)
60 axes.set_xlabel(r'$U$ (eV)', size=10)
61 axes.set_ylabel(r'$E_{U}$ (eV)', size=10)
62 axes.plot(U_Fe_atom_QE, E_Fe_atom_QE, marker='o', ls='none', c='gray')
63 axes.plot(U_Fe_atom_GBRV, E_Fe_atom_GBRV, marker='s', ls='none', c='k')
64 axes.set_xlim(-1, 7)
65 axes.set_xticks([0, 2, 4, 6])
66
67 axes.text(1, 2, 'Fe')
68
69 axes = fig.add_axes([0.51, 0.15, 0.3, 0.4])
70 U_Co_atom_QE, E_Co_atom_QE = np.array(zip(*Co_atom_QE))
71
72 U_Co_atom_GBRV, E_Co_atom_GBRV = np.array(zip(*Co_atom_GBRV))
73
74 E_Co_atom_QE -= E_Co_atom_QE[0]
75 E_Co_atom_GBRV -= E_Co_atom_GBRV[0]

```



```

76
77 axes.plot(U_Co_atom_QE, E_Co_atom_QE, marker='o', ls='none', c='gray')
78 axes.plot(U_Co_atom_GBRV, E_Co_atom_GBRV, marker='s', ls='none', c='k')
79 axes.set_xlim(-1, 7)
80 axes.set_xticks([0, 2, 4, 6])
81 axes.set_ylim(-0.1, 1.1)
82 axes.set_yticklabels([])
83 axes.set_xlabel(r'$U$ (eV)', size=10)
84 axes.text(1, 0.8, 'Co')
85
86 axes = axes.twinx()
87 axes.set_ylim(-0.1, 1.1)
88 axes.set_ylabel(r'$E_U$ (eV)', size=10)
89
90 plt.savefig('supporting-figures/EvsU-atoms.png')
91
92 plt.show()

```

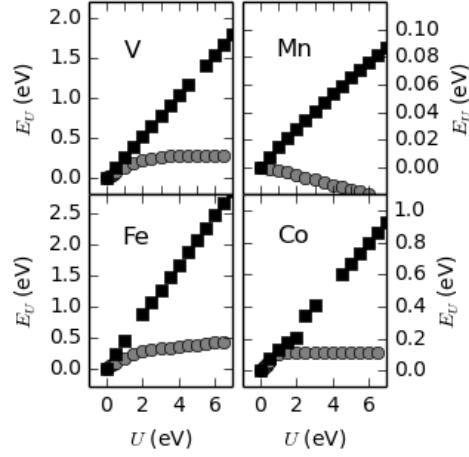


FIG. 1. The pseudopotential (PP) dependent contribution of the Hubbard U to the total energy (E_U) of atomic V, Mn, Fe, and Co. The projections of the d -orbital is defined by the atomic pseudo-wave-functions, which is determined by the PP used. Gray circles are results from the original QE library of PPs and black squares are results from the GBRV high-throughput PPs.

IV. BAND GAPS WITH LINEAR RESPONSE U

The table below lists the calculated and experimental bandgaps along with their citations. The code directly below constructs Figure 2 in the manuscript.

TABLE III. Calculated and experimental band gaps. All units are in eV

Compound	GBRV	GBRV + U	QE	QE + U	Exp	Source
VO	0.57	3.14	0.49	2.92	0.5	[3]
VO2	0	0	0	0	0	[4]
V2O3	0	1.11	0	1.1	1.51	[5]
V2O5	1.84	2.4	1.47	2.06	2.34	[6]
Cr2O3	1.01	3.04	1.13	2.15	2.80	[5]
CrO3	1.95	2.15	1.95	2.03	2.25	[7]
MnO	1.21	3.09	1.5	3.22	3.9	[6]
Mn3O4	0.32	0.87	0.27	0.51	2.07	[8]
MnO2	0	0	0	0	0	[6]
FeO	0	1.9	0	1.8	2.4	[9]
Fe3O4	0	0	0	0	0.25	[10]
Fe2O3	0.5	2.27	0.61	1.46	2.34	[6]
CoO	0	2.52	0	1.7	2.4	[11]
Co3O4	0.1	2.44	0.16	1.5	1.6	[12]

```

1  import matplotlib.pyplot as plt
2  from matplotlib.pyplot import rc, rcParams
3  from matplotlib.patches import Ellipse
4  import numpy as np
5
6  rc('xtick', labelsizes=10)
7  rc('ytick', labelsizes=10)
8
9  GBRV, GBRV_U, QE, QE_U, EXP = [], [], [], [], []
10
11  for name, gbrv, gbrv_u, qe, qe_u, exp, source in data:
12      GBRV.append(gbrv)
13      GBRV_U.append(gbrv_u)
14      QE.append(qe)
15      QE_U.append(qe_u)
16      EXP.append(exp)
17
18  GBRV_MAE = np.mean(np.absolute(np.array(GBRV) - np.array(EXP)))
19  GBRV_U_MAE = np.mean(np.absolute(np.array(GBRV_U) - np.array(EXP)))
20  QE_MAE = np.mean(np.absolute(np.array(QE) - np.array(EXP)))

```

```

21 QE_U_MAE = np.mean(np.absolute(np.array(QE_U) - np.array(EXP)))
22
23 print (GBRV_U_MAE - GBRV_MAE) / GBRV_MAE
24 print (QE_U_MAE - QE_MAE) / QE_MAE
25
26 e_V0 = Ellipse(xy=(1.815, 0.5), width=3.2, height=0.5, angle=0, fill=False)
27 e_Mn304 = Ellipse(xy=(0.57, 2.07), width=1, height=0.3, angle=0, fill=False)
28
29 fig = plt.figure(1, (2.5, 2.5), dpi=300)
30 ax = fig.add_axes([0.2, 0.22, 0.7, 0.7])
31 ax.add_artist(e_V0)
32 ax.add_artist(e_Mn304)
33 ax.annotate(' ', xy=(1.815, 0.75), xytext=(2.2, 1.1), arrowprops=dict(arrowstyle='->'))
34 ax.text(2.2, 1.1, 'V0', size='small')
35 ax.annotate(' ', xy=(0.57, 2), xytext=(0.57, 1.4), arrowprops=dict(arrowstyle='->'))
36 ax.text(0.57, 1.2, r'$\mathdefault{\text{Mn}_{3}\text{O}_{4}}$', ha='center', size='small')
37 ax.plot((0, 4), (0, 4), ls='--', c='k')
38 ax.plot(QE, EXP, marker='s', c='b', ls='none')
39 ax.plot(QE_U, EXP, marker='s', c='r', ls='none')
40 ax.plot(GBRV, EXP, marker='o', c='b', ls='none')
41 ax.plot(GBRV_U, EXP, marker='o', c='r', ls='none')
42 ax.set_xlim(0, 4)
43 ax.set_ylim(0, 4)
44 ax.set_xticks([0, 1, 2, 3, 4])
45 ax.set_yticks([0, 1, 2, 3, 4])
46 ax.set_xlabel('$E_{\text{g}}^{\text{calc}}$ (eV)')
47 ax.set_ylabel('$E_{\text{g}}^{\text{exp}}$ (eV)')
48 plt.savefig('figures/FIG2.png', dpi=300)
49 plt.savefig('figures/FIG2.eps', dpi=300)
50 plt.show()

```

-0.574705882353

-0.478545887962

V. DFT+ $U(V)$ FOR BULK COBALT MONOXIDE

Below lists all of the data needed for the calculation of the DFT+ $U(V)$ curve for CoO. This includes three sets of data. One, we need the calculated, linear response U value at isotropically expanding unit cells. This is shown in Table IV. With these linear response U values, we fit a line to estimate the $U(V)$ behavior and calculate the pressure (eV/Å³) at isotropically expanding unit cells with the U applied from the $U(V)$ relationship we just calculated along with stresses at the $U=0$ and $U=U_{\text{calc}}$ (Tables V – VII). Finally, we

integrate over the negative pressures to calculate the potential energy landscape. The code for extracting the data out of these tables and generating Figure 4 in the manuscript is shown following the tables.

TABLE IV. U vs V data for CoO.

Volume ($\text{\AA}^3/\text{atom}$)	U_{calc}
9.25	4.89
13	4.32
20	4.16
30	3.74
45	3.16
60	1.83
80	0.907

TABLE V. Volume vs pressure for DFT+ $U(V)$ method.

Volume ($\text{\AA}^3/\text{atom}$)	Pressure (eV/ \AA^3)
9.25	-0.0422
9.75	0.0287
10.00	0.0519
10.25	0.0721
10.50	0.0893
11.00	0.1163
11.50	0.1355
12.00	0.1487
13.00	0.1629
15.00	0.1696
16.00	0.1704
17.00	0.1688
19.00	0.1604
20.00	0.1547
22.00	0.1425
26.00	0.1182
28.00	0.1071
32.00	0.0730
36.00	0.0571
38.00	0.0479
40.00	0.0430
45.00	0.0314
50.00	0.0262
55.00	0.0214
60.00	0.0171
65.00	0.0116
70.00	0.0092
75.00	0.0084
80.00	0.0061

TABLE VI. Volume vs pressure at $U=0$.

Volume ($\text{\AA}^3/\$atom)$	Pressure (eV/ \AA^3)
9.25	0.005
9.75	0.063
10.00	0.085
10.25	0.105
10.50	0.122
11.00	0.149
11.50	0.168
12.00	0.183
13.00	0.199
14.00	0.204
18.00	0.183
20.00	0.166
22.00	0.149
28.00	0.106
36.00	0.070
38.00	0.063
40.00	0.057
45.00	0.043
50.00	0.028
55.00	0.022
60.00	0.014
65.00	0.012
70.00	0.013
75.00	0.008
80.00	0.006

TABLE VII. Volume vs pressure at $U=4.89$.

Volume ($\text{\AA}^3/\text{atom}$)	Pressure (eV/ \AA^3)
9.25	-0.044
9.75	0.027
10.00	0.050
10.25	0.070
10.50	0.087
11.00	0.113
11.50	0.132
13.00	0.158
15.00	0.168
17.00	0.167
18.00	0.164
20.00	0.154
28.00	0.082
50.00	0.024
55.00	0.019
60.00	0.017
65.00	0.014
70.00	0.011
75.00	0.009
80.00	0.007

Below is the code for the creation of Figure 4 in the manuscript.

```

1 import numpy as np
2 import matplotlib
3 import matplotlib.pyplot as plt
4 from scipy.interpolate import interp1d
5 from matplotlib.colors import Normalize
6
7 matplotlib.rc('xtick', labelsizes=10)
8 matplotlib.rc('ytick', labelsizes=10)
9

```

```

10 fig = plt.figure(1, (3.25, 5), dpi=300)
11
12 # First plot the UvsV relationship
13 vols, Us = zip(*UofV)
14 coeffs = np.polyfit(vols, Us, 1)
15 UofV = np.poly1d(coeffs)
16
17 ax1 = fig.add_axes((0.2, 0.7, 0.75, 0.28))
18 ax1.plot(vols, Us, marker='s', ls='none', c='k')
19 ax1.plot(vols, UofV(vols), c='k')
20 ax1.set_xticklabels([], [])
21 ax1.set_xlim(5, 80)
22 ax1.set_ylim(0, 6)
23 ax1.set_ylabel('U (eV)', size='small')
24 ax1.text(70, 4.8, '(a)')
25
26 ### Now plot the stresses ###
27
28 U0_vols, U0_Ps = zip(*U0)
29 Ub_vols, Ub_Ps = zip(*Ub)
30 UV_vols, UV_Ps = zip(*UV)
31
32 small_vols = np.linspace(min(UV_vols), 11, 5)
33 medium_vols = np.linspace(11, 15, 4)
34 large_vols = np.linspace(15, max(UV_vols), 15)
35 vols = np.concatenate((small_vols, medium_vols, large_vols))
36
37 f0 = interp1d(U0_vols, U0_Ps, kind='cubic', bounds_error=False)
38 fb = interp1d(Ub_vols, Ub_Ps, kind='cubic', bounds_error=False)
39 fV = interp1d(UV_vols, UV_Ps, kind='cubic', bounds_error=False)
40
41 # Normalization for the color scheme
42 norm = Normalize(vmin=UofV(max(vols)), vmax=UofV(min(vols)))
43 cm = plt.get_cmap('jet')
44 color_max = cm(0.99)
45 color_min = cm(0)
46
47 ax2 = fig.add_axes((0.2, 0.4, 0.75, 0.28))
48 ax2.plot(vols, f0(vols), c=color_min, lw=1.5, zorder=1)
49 ax2.plot(vols, fb(vols), c=color_max, lw=1.5, zorder=1)
50 ax2.scatter(vols, fV(vols), c=UofV(vols), s=40, cmap=plt.get_cmap('jet'), norm=norm, zorder=2,
51            marker='^')
52 ax2.set_xticklabels([], [])
53 ax2.set_xlim(5, 80)
54 ax2.set_ylim(-0.02, 0.22)
55 ax2.set_ylabel(r'Pressure (eV/$\AA^3$)', size='small')
56 ax2.text(70, 0.17, '(b)')

```



```

57
58 ### Now plot the total energies ###
59
60 vols = np.linspace(min(U0_vols), max(U0_vols), 20)
61
62 U0_Es, Ub_Es, UV_Es = [0], [0], [0]
63 for vol in vols[1:]:
64     int_vols = np.linspace(min(vols), vol)
65     U0_E = np.trapz(f0(int_vols), int_vols)
66     Ub_E = np.trapz(fb(int_vols), int_vols)
67     UV_E = np.trapz(fV(int_vols), int_vols)
68     U0_Es.append(U0_E)
69     Ub_Es.append(Ub_E)
70     UV_Es.append(UV_E)
71
72 ax3 = fig.add_axes((0.2, 0.1, 0.75, 0.28))
73
74 ax3.plot(vols, U0_Es, c=color_min, lw=1.5, zorder=1)
75 ax3.plot(vols, Ub_Es, c=color_max, lw=1.5, zorder=1)
76 ax3.scatter(vols, UV_Es, c=UofV(vols), s=40, cmap=plt.get_cmap('jet'), norm=norm, zorder=2)
77 ax3.set_xlim(5, 80)
78 ax3.set_ylim(0, 5)
79 ax3.set_xlabel(r'Volume ( $\text{\AA}^3/\text{atom}$ )', size='small')
80 ax3.set_ylabel('Relative Energy (eV)', size='small')
81 ax3.text(10, 4, '(c)')
82
83 # Print the x value of the CoO system
84 x = (UV_Es[-1] - U0_Es[-1]) / (Ub_Es[-1] - U0_Es[-1])
85 print 'The x value of the CoO system is {x:1.3f}'.format(**locals())
86
87 # Plot the color legend
88 gradient = np.linspace(0, 1, 256)
89 gradient = np.vstack((gradient, gradient))
90 ax4 = fig.add_axes((0.7, 0.52, 0.2, 0.03))
91 ax4.imshow(gradient, aspect='auto', cmap=plt.get_cmap('jet'))
92 ax4.set_yticks([], [])
93 ax4.set_xticks((0, 256))
94 ax4.set_xticklabels((0, 4.89))
95 fig.text(0.787, 0.56, 'U', size=10, style='italic')
96 fig.savefig('figures/FIG4.eps', dpi=300)
97 fig.savefig('figures/FIG4.png', dpi=300)
98 plt.show()

```

VI. TABLES DATA NEEDED FOR DFT+ $U(V)$ ENERGY

The tables below contains the information needed to calculate the DFT+ $U(V)$ energy. With the fitted x parameter, the DFT+ $U(V)$ energy is simplified to

$$\Delta E_{\text{DFT}+U(V)} = x\Delta E_{\text{coh}}^{U=U_{\text{calc}}} + (1-x)\Delta E_{\text{coh}}^{U=U_0}, \quad (3)$$

where $\Delta E_{\text{coh}}^{U=U_a}$ is cohesive energy with a U_a value of either the linear response calculated U value (U_{calc}) or $U = 0$. With respect to isolated metal atoms and molecular oxygen, the cohesive energy is defined by,

$$\Delta E_{\text{coh}}^{U=U_a} = E_{\text{MO}_y}^{U=U_a} - E_{\text{M}_{\text{atom}}}^{U=U_a} - \frac{y}{2}E_{\text{O}_2}. \quad (4)$$

From Equation 3 and 4, we see that the ingredients we need for the DFT+ $U(V)$ energy for a single material is the total energy of the single material at both the calculated linear response U value and $U = 0$ along with the total energies of the transition metal as an isolated atom at those U values. The tables below contains the data of the materials in the bulk phase along the calculated linear response U value, stoichiometry, and experimental formation energy. $E(U)$ of the atomic species required for the cohesive energy in Equation 4 is shown in Section XI and obtained for specific U values using a spline fit through `numpy`.

TABLE VIII. Table of total enery data required for calculating the DFT+ U (V) energy with a fitted x value from the original QUANTUM-ESPRESSO library. E_{U0} and E_{Ucalc} are the total energies at $U = 0$ and $U = U_{calc}$ normalized to per metal atom. U_{calc} is the linear response calculted U value of the bulk material. O_{frac} is the amount of oxygen in each compound normalized to one metal atom. E_{form}^{exp} is the experimental formation energy, also normalized per metal atom, obtained from the Kubaschweski tables [13]. All energy values are in eV.

Compound	E_{U0}	E_{Ucalc}	U_{calc}	O_{frac}	E_{form}^{exp}
VO	-811.841	-809.196	4.12	1	-4.475
V ₂ O ₃	-1030.252	-1025.794	4.99	$\frac{3}{2}$	-6.346
VO ₂	-1247.732	-1242.103	5.14	2	-7.39802
V ₂ O ₅	-1464.720	-1458.494	5.12	$\frac{5}{2}$	-8.033
Cr ₂ O ₃	-3021.814	-3019.755	2.73	$\frac{3}{2}$	-5.88017
CrO ₃	-3671.592	-3665.638	4.99	3	-6.084
MnO	-3282.408	-3281.177	4.94	1	-3.989
Mn ₃ O ₄	-3427.593	-3425.846	4.05	$\frac{4}{3}$	-4.793
MnO ₂	-3716.734	-3713.366	4.78	2	-5.39875
FeO	-1193.307	-1191.480	4.10	1	-2.72
Fe ₃ O ₄	-1338.562	-1336.387	3.72	$\frac{4}{3}$	-3.96
Fe ₂ O ₃	-1410.831	-1408.883	3.47	$\frac{3}{2}$	-4.35
CoO	-1445.944	-1444.136	4.72	1	-2.464
Co ₃ O ₄	-1590.986	-1588.436	5.43	$\frac{4}{3}$	-3.144

TABLE IX. Table of total energy data required for calculating the DFT+ $U(V)$ energy with a fitted x value from the GBRV high-throughput pseudopotential library. E_{U0} and E_{Ucalc} are the total energies at $U = 0$ and $U = U_{calc}$ normalized to per metal atom. U_{calc} is the linear response calculated U value of the bulk material. O_{frac} is the amount of oxygen in each compound normalized to one metal atom. E_{form}^{exp} is the experimental formation energy, also normalized per metal atom, obtained from the Kubaschewski tables [13]. All energy values are in eV.

Compound	E_{U0}	E_{Ucalc}	U_{calc}	O_{frac}	E_{form}^{exp}
VO	-2414.890	-2412.487	3.61	1	-4.475
V ₂ O ₃	-2635.916	-2631.853	4.64	$\frac{3}{2}$	-6.346
VO ₂	-2856.144	-2850.141	5.72	2	-7.39802
V ₂ O ₅	-3075.806	-3070.138	4.67	$\frac{5}{2}$	-8.033
Cr ₂ O ₃	-3049.517	-3046.607	4.86	$\frac{3}{2}$	-5.88017
CrO ₃	-3707.472	-3698.280	7.42	3	-6.084
MnO	-3323.890	-3322.734	5.52	1	-3.989
Mn ₃ O ₄	-3470.900	-3468.568	6.11	$\frac{4}{3}$	-4.793
MnO ₂	-3763.646	-3759.349	6.63	2	-5.39875
FeO	-3888.019	-3885.982	5.80	1	-2.72
Fe ₃ O ₄	-4035.104	-4032.264	6.07	$\frac{4}{3}$	-3.96
Fe ₂ O ₃	-4108.287	-4105.977	5.21	$\frac{3}{2}$	-4.35
CoO	-4499.872	-4498.105	5.86	1	-2.464
Co ₃ O ₄	-4646.862	-4643.617	7.44	$\frac{4}{3}$	-3.144

VII. DFT+ $U(V)$ OXIDATION ENERGIES OF OXIDES

Below lists all the code for performing the data analysis. The reactions used are shown in the table below.

TABLE X. A list of all oxidation reactions we tested in this study. The training set (reactions with V and Fe) are the reactions we used to fit the single empirical parameter, x , which allows for a faster evaluation of the DFT+ $U(V)$ reference energy. The test set (reactions with Mn and Co) are the reactions we used to test this single parameter, x , and the predictive power of our approach. All experimental formation enthalpies used for reaction enthalpies were taken from the Kubaschewski tables [13].

Oxidation Reactions
$\text{VO} + \frac{1}{4} \text{O}_2 \rightarrow \frac{1}{2} \text{V}_2\text{O}_3$
$\text{VO} + \frac{1}{2} \text{O}_2 \rightarrow \text{VO}_2$
$\text{VO} + \frac{3}{4} \text{O}_2 \rightarrow \frac{1}{2} \text{V}_2\text{O}_5$
$\frac{1}{2} \text{V}_2\text{O}_3 + \frac{1}{4} \text{O}_2 \rightarrow \text{VO}_2$
$\frac{1}{2} \text{V}_2\text{O}_3 + \frac{1}{2} \text{O}_2 \rightarrow \frac{1}{2} \text{V}_2\text{O}_5$
$\text{VO}_2 + \frac{1}{4} \text{O}_2 \rightarrow \frac{1}{2} \text{V}_2\text{O}_5$
$\text{MnO} + \frac{1}{6} \text{O}_2 \rightarrow \frac{1}{3} \text{Mn}_3\text{O}_4$
$\text{MnO} + \frac{1}{2} \text{O}_2 \rightarrow \text{MnO}_2$
$\frac{1}{3} \text{Mn}_3\text{O}_4 + \frac{1}{3} \text{O}_2 \rightarrow \text{MnO}_2$
$\text{FeO} + \frac{1}{6} \text{O}_2 \rightarrow \frac{1}{3} \text{Fe}_3\text{O}_4$
$\text{FeO} + \frac{1}{4} \text{O}_2 \rightarrow \frac{1}{2} \text{Fe}_2\text{O}_3$
$\frac{1}{3} \text{Fe}_3\text{O}_4 + \frac{1}{12} \text{O}_2 \rightarrow \frac{1}{2} \text{Fe}_2\text{O}_3$
$\text{CoO} + \frac{1}{6} \text{O}_2 \rightarrow \frac{1}{3} \text{Co}_3\text{O}_4$

As said in the manuscript, the DFT+ $U(V)$ energy of each material can be characterized by a material specific weighting factor x and can be fit by minimizing errors between calculated and experimental oxidation energies. In the following sections, we first present code for calculating the x value that minimizes the errors of the training set reactions. We then compile the reaction energies of both the training and test set using this x value.

We then present code for calculating the x value that minimizes the errors in both the training and test set of reactions as well as the 95% confidence intervals around x . We do this using the `pycse.py` module created by John Kitchin, which is detailed in the Section XII. We then calculate the errors in oxidation energies produced by using the optimal x as well as the 95% confidence intervals in the oxidation energies produced by the 95% confidence

intervals on x .

A. Fitting the x parameter to experimental data

The code below performs simple linear regression to find the x value that minimizes errors between calculated and experimental oxidation energies for both training reactions and all reactions, which is needed for Figures 5 (a) and (b). We also desire the 95% confidence intervals on x as well. To do this, we must recast our problem as a linear equation where x is the parameter we want to find and use the `nlinfit` function found in the `pycse` module. Note, we use `nlinfit` instead of `regress` because we are fixing the y -intercept.

The set of equations we want to fit are oxidation energies. Written in terms of the DFT+ $U(V)$ energy, the oxidation energy is given by

$$\Delta H_{rxn}^{exp} = \Delta E_{\text{DFT}+U(\mathbf{V}),\text{MO}_y} - \Delta E_{\text{DFT}+U(\mathbf{V}),\text{MO}_x}, \quad (5)$$

where ΔH_{rxn}^{exp} is known. However, the x value is buried in the $\Delta E_{\text{DFT}+U(\mathbf{V})}$ energies. After expanding this term out, we get

$$\Delta H_{rxn}^{exp} = (x\Delta E_{\text{coh},\text{MO}_y}^{U=U_{\text{calc}}} + (1-x)\Delta E_{\text{coh},\text{MO}_y}^{U=U_0}) - (x\Delta E_{\text{coh},\text{MO}_x}^{U=U_{\text{calc}}} + (1-x)\Delta E_{\text{coh},\text{MO}_x}^{U=U_0}). \quad (6)$$

We recast Equation (6) into an equation that's clearly linear with x ,

$$\begin{aligned} \Delta H_{rxn}^{exp} - \Delta E_{\text{coh},\text{MO}_y}^{U=U_0} + \Delta E_{\text{coh},\text{MO}_x}^{U=U_0} = x[& (\Delta E_{\text{coh},\text{MO}_y}^{U=U_{\text{calc}}} - \Delta E_{\text{coh},\text{MO}_y}^{U=U_0}) \\ & - (\Delta E_{\text{coh},\text{MO}_x}^{U=U_{\text{calc}}} - \Delta E_{\text{coh},\text{MO}_x}^{U=U_0})], \end{aligned} \quad (7)$$

where Equation (7) can thought simply as a $a = xb$ linear equation where a and b are known for each oxidation energy we want to fit from either experimental data or DFT calculations and x is the parameter we wish to find. The two sets of code below calculates fits the x value to the training reactions and all reactions. Note we only use the confidence intervals in our analysis of fitting x to all the reactions.

```

1 import sys
2 import numpy as np
3 from scipy.stats.distributions import t

```

```

4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn', 'Cr')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)
37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41 for element in elements:
42     atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                         atom_data_gbrv[element]['E'], kind='cubic')
44
45 ## Now calculate the oxidation energies using each pseudopotential
46 RXNS = [('V205', 'V0'),      # r1: V0 + 3/4 O2 = V0_2.5
47          ('V02', 'V0'),      # r2: V0 + 1/2 O2 = V0_2
48          ('V203', 'V0'),      # r3: V0 + 1/4 O2 = V0_1.5
49          ('V205', 'V203'),    # r4: V0_1.5 + 1/2 O2 = V0_2.5
50          ('V02', 'V203'),     # r5: V0_1.5 + 1/4 O2 = V0_2

```

```

51         ('V205', 'V02'),      # r6:  $VO_2 + 1/4 O_2 = VO_{2.5}$ 
52         ('Cr03', 'Cr203'),    # r7:  $CrO_{1.5} + 3/4 O_2 = CrO_3$ 
53         ('Fe203', 'FeO'),     # r10:  $FeO + 1/4 O_2 = FeO_{1.5}$ 
54         ('Fe304', 'FeO'),     # r11:  $FeO + 1/6 O_2 = FeO_{1.33}$ 
55         ('Fe203', 'Fe304'),   # r12:  $FeO_{1.33} + 1/12 O_2 = FeO_{1.5}$ 
56     ]
57
58     def calc_error(atom_data, rxns, bulk_data, O2):
59         bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
60         O2 += 1.27580582171
61         ys, Es = [], []
62         Ubcohs, UOcohs = {}, {}
63         cohavgs, Ofrac, tot_energy, enthalpies = {}, {}, {}, {}
64         for atoms, E0, EUcalc, Ucalc, O_frac, exp in bulk_data:
65             element = atoms.translate(None, '00123456789')
66             bulkUcoh = EUcalc - atom_data[element](Ucalc) - O_frac * O2 / 2
67             atomUcoh = E0 - atom_data[element](0) - O_frac * O2 / 2
68
69             enthalpies[atoms] = exp
70             Ubcohs[atoms] = bulkUcoh
71             UOcohs[atoms] = atomUcoh
72
73         for pair in rxns:
74             y = (enthalpies[pair[0]] - enthalpies[pair[1]]) - (UOcohs[pair[0]] - UOcohs[pair[1]])
75             E = (Ubcohs[pair[0]] - UOcohs[pair[0]]) - (Ubcohs[pair[1]] - UOcohs[pair[1]])
76             ys.append(y)
77             Es.append(E)
78         return ys, Es
79
80     ys, Es = calc_error(atom_fits_qe, RXNS, qe_data, -864.185)
81     ys = np.array(ys)
82     Es = np.array(Es)
83
84     def model(E, p):
85         return E*p
86
87     [xfit], [xint], SE = nlinfit(model, Es, ys, 0.57, alpha=0.05)
88
89     print 'The x that minimizes the error is {xfit:1.3f} for the training reactions using the QE PPs'.format(**locals())
90     print ''
91
92     ys, Es = calc_error(atom_fits_gbrv, RXNS, gbrv_data, -874.942962698)
93     ys = np.array(ys)
94     Es = np.array(Es)
95
96     def model(E, p):
97         return E*p

```



```

98
99 [xfit], [xint], SE = nlinfit(model, Es, ys, 0.57, alpha=0.05)
100
101 print 'The x that minimizes the error is {xfit:1.3f} for the training reactions using the GBRV PPs'.format(**locals())

```

The x that minimizes the error is 0.589 for the training reactions using the QE PPs

The x that minimizes the error is 0.486 for the training reactions using the GBRV PPs

```

1  import sys
2  import numpy as np
3  from scipy.stats.distributions import t
4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn', 'Cr')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)

```

```

37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41     for element in elements:
42         atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                             atom_data_gbrv[element]['E'], kind='cubic')
44
45     ## Now calculate the oxidation energies using each pseudopotential
46     RXNS = [('V205', 'V0'),      # r1:  $VO + 3/4 O2 = VO_{2.5}$ 
47             ('V02', 'V0'),      # r2:  $VO + 1/2 O2 = VO_2$ 
48             ('V203', 'V0'),      # r3:  $VO + 1/4 O2 = VO_{1.5}$ 
49             ('V205', 'V203'),    # r4:  $VO_{1.5} + 1/2 O2 = VO_{2.5}$ 
50             ('V02', 'V203'),    # r5:  $VO_{1.5} + 1/4 O2 = VO_2$ 
51             ('V205', 'V02'),    # r6:  $VO_2 + 1/4 O2 = VO_{2.5}$ 
52             ('Cr03', 'Cr203'),   # r7:  $CrO_{1.5} + 3/4 O2 = CrO_3$ 
53             ('Mn02', 'Mn0'),    # r8:  $MnO + 1/2 O2 = MnO_2$ 
54             ('Mn304', 'Mn0'),    # r9:  $MnO + 1/6 O2 = MnO_{1.33}$ 
55             ('Mn02', 'Mn304'),   # r10:  $MnO_{1.33} + 1/3 O2 = MnO_2$ 
56             ('Fe203', 'Fe0'),    # r11:  $FeO + 1/4 O2 = FeO_{1.5}$ 
57             ('Fe304', 'Fe0'),    # r12:  $FeO + 1/6 O2 = FeO_{1.33}$ 
58             ('Fe203', 'Fe304'),  # r13:  $FeO_{1.33} + 1/12 O2 = FeO_{1.5}$ 
59             ('Co304', 'Co0'),    # r14:  $CoO + 1/6 O2 = CoO_{1.33}$ 
60     ]
61
62     def calc_error(atom_data, rxns, bulk_data, O2):
63         bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
64         O2 += 1.27580582171
65         ys, Es = [], []
66         Ubcohs, U0cohs = {}, {}
67         cohavgs, Ofrac, tot_energy, enthalpies = {}, {}, {}, {}
68         for atoms, E0, EUcalc, Ucalc, O_frac, exp in bulk_data:
69             element = atoms.translate(None, '00123456789')
70             bulkUcoh = EUcalc - atom_data[element](Ucalc) - O_frac * O2 / 2
71             atomUcoh = E0 - atom_data[element](0) - O_frac * O2 / 2
72
73             enthalpies[atoms] = exp
74             Ubcohs[atoms] = bulkUcoh
75             U0cohs[atoms] = atomUcoh
76
77         for pair in rxns:
78             y = (enthalpies[pair[0]] - enthalpies[pair[1]]) - (U0cohs[pair[0]] - U0cohs[pair[1]])
79             E = (Ubcohs[pair[0]] - U0cohs[pair[0]]) - (Ubcohs[pair[1]] - U0cohs[pair[1]])
80             ys.append(y)
81             Es.append(E)
82         return ys, Es
83

```

```

84  ys, Es = calc_error(atom_fits_qe, RXNS, qe_data, -864.185)
85  ys = np.array(ys)
86  Es = np.array(Es)
87
88  def model(E, p):
89      return E*p
90
91  [xfit], [xint], SE = nlinfit(model, Es, ys, 0.57, alpha=0.05)
92
93  print 'The x that minimizes the error is {xfit:1.3f} for all reactions using the QE PPs'.format(**locals())
94  print 'The 95% confidence intervals on x are {xint[0]:1.3f} and {xint[1]:1.3f}'.format(**locals())
95  print ''
96
97  ys, Es = calc_error(atom_fits_gbrv, RXNS, gbrv_data, -874.942962698)
98  ys = np.array(ys)
99  Es = np.array(Es)
100
101  def model(E, p):
102      return E*p
103
104  [xfit], [xint], SE = nlinfit(model, Es, ys, 0.57, alpha=0.05)
105
106  print 'The x that minimizes the error is {xfit:1.3f} for all reactions using the GBRV PPs'.format(**locals())
107  print 'The 95% confidence intervals on x are {xint[0]:1.3f} and {xint[1]:1.3f}'.format(**locals())

```

The x that minimizes the error is 0.599 for all reactions using the QE PPs

The 95% confidence intervals on x are 0.546 and 0.652

The x that minimizes the error is 0.487 for all reactions using the GBRV PPs

The 95% confidence intervals on x are 0.407 and 0.567

B. Store test/training set oxidation energies in table

The code below takes the x value fitted to the training reactions, calculates the oxidation energies of both the training set and test set with this x , and finds the MAE of both sets compared to experimental values. It does this for both sets of pseudopotentials. The tabular output is shown following the code.

```

1  import sys
2  import numpy as np
3  from scipy.stats.distributions import t

```

```

4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn', 'Cr')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)
37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41 for element in elements:
42     atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                         atom_data_gbrv[element]['E'], kind='cubic')
44
45 ## Now calculate the oxidation energies using each pseudopotential
46 TRAIN_RXNS = [('V205', 'VO'),      # r1: VO + 3/4 O2 = VO_2.5
47                ('V02', 'VO'),      # r2: VO + 1/2 O2 = VO_2
48                ('V203', 'VO'),      # r3: VO + 1/4 O2 = VO_1.5
49                ('V205', 'V203'),    # r4: VO_1.5 + 1/2 O2 = VO_2.5
50                ('V02', 'V203'),     # r5: VO_1.5 + 1/4 O2 = VO_2

```

```

51         ('V2O5', 'V02'),      # r6:  $VO_2 + 1/4 O_2 = VO_{2.5}$ 
52         ('CrO3', 'Cr2O3'),    # r7:  $CrO_{1.5} + 3/4 O_2 = CrO_3$ 
53         ('Fe2O3', 'FeO'),     # r11:  $FeO + 1/4 O_2 = FeO_{1.5}$ 
54         ('Fe3O4', 'FeO'),     # r12:  $FeO + 1/6 O_2 = FeO_{1.33}$ 
55         ('Fe2O3', 'Fe3O4'),   # r13:  $FeO_{1.33} + 1/12 O_2 = FeO_{1.5}$ 
56     ]
57
58     TEST_RXNS = [('MnO2', 'MnO'),      # r8:  $MnO + 1/2 O_2 = MnO_2$ 
59                 ('Mn3O4', 'MnO'),    # r9:  $MnO + 1/6 O_2 = MnO_{1.33}$ 
60                 ('MnO2', 'Mn3O4'),   # r10:  $MnO_{1.33} + 1/3 O_2 = MnO_2$ 
61                 ('Co3O4', 'CoO'),    # r14:  $CoO + 1/6 O_2 = CoO_{1.33}$ 
62     ]
63
64     def calc_error(x, atom_data, rxns, bulk_data, O2):
65         bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
66         O2 += 1.27580582171
67         ys, Es = [], []
68         Ubcohs, UOcohs = {}, {}
69         cohavgs, cohatom, Ofrac, enthalpies = {}, {}, {}, {}
70         for atoms, EO, EUcalc, Ucalc, O_frac, exp in bulk_data:
71             element = atoms.translate(None, '00123456789')
72             bulkUcoh = EUcalc - atom_data[element](Ucalc) - O_frac * O2 / 2
73             atomUcoh = EO - atom_data[element](0) - O_frac * O2 / 2
74             avgUcoh = x * bulkUcoh + (1 - x) * atomUcoh
75             cohavgs[atoms] = avgUcoh
76             cohatom[atoms] = atomUcoh
77             Ofrac[atoms] = O_frac
78             enthalpies[atoms] = exp
79
80         dft_errors, uv_errors, exp_oxis = [], [], []
81         for pair in rxns:
82             s = '{0}|{1:1.3f}|{2:1.3f}|'
83             uv = cohavgs[pair[0]] - cohavgs[pair[1]]
84             exp = enthalpies[pair[0]] - enthalpies[pair[1]]
85             dft = cohatom[pair[0]] - cohatom[pair[1]]
86             uv_errors.append(abs(exp - uv))
87             rxn = ''
88             for letter in pair[1]:
89                 if letter.isdigit():
90                     rxn += '_{' + letter + '}'
91                 else:
92                     rxn += letter
93             rxn += ' $\rightarrow$ '
94             for letter in pair[0]:
95                 if letter.isdigit():
96                     rxn += '_{' + letter + '}'
97                 else:

```

```

98         rxn += letter
99         print s.format(rxn, exp, uv)
100
101     return np.average(uv_errors)
102
103     print '##CAPTION: Oxidation energies of training set with QE PPs.'
104     print '##ATTR_LATEX: :align c|c|c :placement [H]'
105     print '##TBLNAME: train-QE'
106     print '|Reaction|E_{exp}|E_{DFT+\\textit{U}(\\V)}|'
107     print '|--|'
108
109     error = calc_error(0.589, atom_fits_qe, TRAIN_RXNS, qe_data, -864.185)
110
111     print ''
112
113     print 'QE with training set MAE: {0:1.3f} eV\\n'.format(error)
114
115     print '##CAPTION: Oxidation energies of test set with QE PPs.'
116     print '##ATTR_LATEX: :align c|c|c :placement [H]'
117     print '##TBLNAME: test-QE'
118     print '|Reaction|E_{exp}|E_{DFT+\\textit{U}(\\V)}|'
119     print '|--|'
120
121     error = calc_error(0.589, atom_fits_qe, TEST_RXNS, qe_data, -864.185)
122
123     print ''
124
125     print 'QE with test set MAE: {0:1.3f} eV\\n'.format(error)
126
127     print '##CAPTION: Oxidation energies of training set with GBRV PPs.'
128     print '##ATTR_LATEX: :align c|c|c :placement [H]'
129     print '##TBLNAME: train-GBRV'
130     print '|Reaction|E_{exp}|E_{DFT+\\textit{U}(\\V)}|'
131     print '|--|'
132
133     error = calc_error(0.486, atom_fits_gbrv, TRAIN_RXNS, gbrv_data, -874.942962698)
134
135     print ''
136
137     print 'GBRV with training set MAE: {0:1.3f} eV\\n'.format(error)
138
139     print '##CAPTION: Oxidation energies of test set with GBRV PPs.'
140     print '##ATTR_LATEX: :align c|c|c :placement [H]'
141     print '##TBLNAME: test-GBRV'
142     print '|Reaction|E_{exp}|E_{DFT+\\textit{U}(\\V)}|'
143     print '|--|'
144

```

```

145 error = calc_error(0.486, atom_fits_gbrv, TEST_RXNS, gbrv_data, -874.942962698)
146
147 print ''
148
149 print 'GBRV with test set MAE: {0:1.3f} eV\n'.format(error)

```

TABLE XI. Oxidation energies of training set with QE PPs.

Reaction	E_{exp}	$E_{\text{DFT}+U(\text{V})}$
$\text{VO} \rightarrow \text{V}_2\text{O}_5$	-3.558	-3.591
$\text{VO} \rightarrow \text{VO}_2$	-2.923	-2.682
$\text{VO} \rightarrow \text{V}_2\text{O}_3$	-1.871	-1.619
$\text{V}_2\text{O}_3 \rightarrow \text{V}_2\text{O}_5$	-1.687	-1.972
$\text{V}_2\text{O}_3 \rightarrow \text{VO}_2$	-1.052	-1.064
$\text{VO}_2 \rightarrow \text{V}_2\text{O}_5$	-0.635	-0.909
$\text{Cr}_2\text{O}_3 \rightarrow \text{CrO}_3$	-0.204	-0.298
$\text{FeO} \rightarrow \text{Fe}_2\text{O}_3$	-1.630	-1.713
$\text{FeO} \rightarrow \text{Fe}_3\text{O}_4$	-1.240	-1.224
$\text{Fe}_3\text{O}_4 \rightarrow \text{Fe}_2\text{O}_3$	-0.390	-0.489

QE with training set MAE: 0.139 eV

TABLE XII. Oxidation energies of test set with QE PPs.

Reaction	E_{exp}	$E_{\text{DFT}+U(\text{V})}$
$\text{MnO} \rightarrow \text{MnO}_2$	-1.410	-1.613
$\text{MnO} \rightarrow \text{Mn}_3\text{O}_4$	-0.804	-1.064
$\text{Mn}_3\text{O}_4 \rightarrow \text{MnO}_2$	-0.606	-0.549
$\text{CoO} \rightarrow \text{Co}_3\text{O}_4$	-0.680	-0.787

QE with test set MAE: 0.157 eV

TABLE XIII. Oxidation energies of training set with GBRV PPs.

Reaction	E_{exp}	$E_{\text{DFT}+U(V)}$
$\text{VO} \rightarrow \text{V}_2\text{O}_5$	-3.558	-4.211
$\text{VO} \rightarrow \text{VO}_2$	-2.923	-2.933
$\text{VO} \rightarrow \text{V}_2\text{O}_3$	-1.871	-1.931
$\text{V}_2\text{O}_3 \rightarrow \text{V}_2\text{O}_5$	-1.687	-2.280
$\text{V}_2\text{O}_3 \rightarrow \text{VO}_2$	-1.052	-1.002
$\text{VO}_2 \rightarrow \text{V}_2\text{O}_5$	-0.635	-1.278
$\text{Cr}_2\text{O}_3 \rightarrow \text{CrO}_3$	-0.204	0.253
$\text{FeO} \rightarrow \text{Fe}_2\text{O}_3$	-1.630	-1.605
$\text{FeO} \rightarrow \text{Fe}_3\text{O}_4$	-1.240	-1.136
$\text{Fe}_3\text{O}_4 \rightarrow \text{Fe}_2\text{O}_3$	-0.390	-0.469

GBRV with training set MAE: 0.267 eV

TABLE XIV. Oxidation energies of test set with GBRV PPs.

Reaction	E_{exp}	$E_{\text{DFT}+U(V)}$
$\text{MnO} \rightarrow \text{MnO}_2$	-1.410	-1.402
$\text{MnO} \rightarrow \text{Mn}_3\text{O}_4$	-0.804	-0.830
$\text{Mn}_3\text{O}_4 \rightarrow \text{MnO}_2$	-0.606	-0.572
$\text{CoO} \rightarrow \text{Co}_3\text{O}_4$	-0.680	-0.761

GBRV with test set MAE: 0.037 eV

C. Store DFT/DFT+ $U(V)$ with confidence intervals in table

The code below takes the x value fitted to all reactions along with the confidence intervals on x , calculates the oxidation energies of both the training set and test set with this x along with the high and low 95% confidence intervals of x , and finds the MAE of both sets compared to experimental values. It does this for both sets of pseudopotentials. The tabular output is shown following the code.


```

1  import sys
2  import numpy as np
3  from scipy.stats.distributions import t
4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn', 'Cr')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)
37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41 for element in elements:
42     atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                         atom_data_gbrv[element]['E'], kind='cubic')
44
45 ## Now calculate the oxidation energies using each pseudopotential
46 RXNS = [('V205', 'VO'),      # r1: VO + 3/4 O2 = VO_2.5
47         ('VO2', 'VO'),       # r2: VO + 1/2 O2 = VO_2

```

```

48         ('V203', 'V0'),          # r3:  $VO + 1/4 O2 = VO_{1.5}$ 
49         ('V205', 'V203'),        # r4:  $VO_{1.5} + 1/2 O2 = VO_{2.5}$ 
50         ('V02', 'V203'),          # r5:  $VO_{1.5} + 1/4 O2 = VO_2$ 
51         ('V205', 'V02'),          # r6:  $VO_2 + 1/4 O2 = VO_{2.5}$ 
52         ('Cr03', 'Cr203'),        # r7:  $CrO_{1.5} + 3/4 O2 = CrO_3$ 
53         ('Mn02', 'Mn0'),          # r8:  $MnO + 1/2 O2 = MnO_2$ 
54         ('Mn304', 'Mn0'),         # r9:  $MnO + 1/6 O2 = MnO_{1.33}$ 
55         ('Mn02', 'Mn304'),        # r10:  $MnO_{1.33} + 1/3 O2 = MnO_2$ 
56         ('Fe203', 'Fe0'),         # r11:  $FeO + 1/4 O2 = FeO_{1.5}$ 
57         ('Fe304', 'Fe0'),         # r12:  $FeO + 1/6 O2 = FeO_{1.33}$ 
58         ('Fe203', 'Fe304'),       # r13:  $FeO_{1.33} + 1/12 O2 = FeO_{1.5}$ 
59         ('Co304', 'Co0'),         # r14:  $CoO + 1/6 O2 = CoO_{1.33}$ 
60     ]
61
62     def calc_error(x, atom_data, rxns, bulk_data, O2, x_low, x_high):
63         bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
64         O2 += 1.27580582171
65         ys, Es = [], []
66         Ubcohs, UOcohs = {}, {}
67         cohavgs, cohatom, Ofrc, enthalpies = {}, {}, {}, {}
68         cohavgs_low, cohavgs_high = {}, {}
69         for atoms, E0, EUcalc, Ucalc, Ofrc, exp in bulk_data:
70             element = atoms.translate(None, '00123456789')
71             bulkUcoh = EUcalc - atom_data[element](Ucalc) - Ofrc * O2 / 2
72             atomUcoh = E0 - atom_data[element](0) - Ofrc * O2 / 2
73
74             cohavgs[atoms] = x * bulkUcoh + (1 - x) * atomUcoh
75             cohavgs_low[atoms] = x_low * bulkUcoh + (1 - x_low) * atomUcoh
76             cohavgs_high[atoms] = x_high * bulkUcoh + (1 - x_high) * atomUcoh
77
78             cohatom[atoms] = atomUcoh
79             Ofrc[atoms] = Ofrc
80             enthalpies[atoms] = exp
81
82         dft_errors, uv_errors, exp_oxis = [], [], []
83         for pair in rxns:
84             s = '{0}|{1:1.3f}|{2:1.3f}|{3:1.3f}|{4:1.3f}|{5:1.3f}|'
85             uv = cohavgs[pair[0]] - cohavgs[pair[1]]
86             uv_low = abs(uv - (cohavgs_low[pair[0]] - cohavgs_low[pair[1]]))
87             uv_high = abs(uv - (cohavgs_high[pair[0]] - cohavgs_high[pair[1]]))
88             exp = enthalpies[pair[0]] - enthalpies[pair[1]]
89             dft = cohatom[pair[0]] - cohatom[pair[1]]
90             dft_errors.append(abs(exp - dft))
91             uv_errors.append(abs(exp - uv))
92             rxn = ''
93             for letter in pair[1]:
94                 if letter.isdigit():

```

```

95         rxn += '_{' + letter + '}'
96     else:
97         rxn += letter
98     rxn += ' $\rightarrow$ '
99     for letter in pair[0]:
100         if letter.isdigit():
101             rxn += '_{' + letter + '}'
102         else:
103             rxn += letter
104
105     print s.format(rxn, exp, dft, uv, uv_low, uv_high)
106
107     return np.average(dft_errors), np.average(uv_errors)
108
109 print '#+CAPTION: All oxidation energies with QE PPs. E_{exp}, E_{DFT} and E_{DFT+\textit{U}(\V)} are the experimental, DFT and
110 print '#+ATTR_LATEX: :placement [H] :align c|c|c|c|c|c'
111 print '#+TBLNAME: all-QE'
112 print '|Reaction|E_{Exp}|E_{DFT}|E_{DFT+\textit{U}(\V)}|- E_{DFT+\textit{U}(\V)}|+ E_{DFT+\textit{U}(\V)}|'
113 print '|--|'
114
115 qe_dft_error, qe_uv_error = calc_error(0.599, atom_fits_qe, RXNS, qe_data, -864.185, 0.546, 0.652)
116
117 print ''
118
119 print 'QE Library: DFT MAE = {0:1.3f}, DFT+U(V) MAE = {1:1.3f}, x=0.599 +- 0.053'.format(qe_dft_error, qe_uv_error)
120 print '{0:1.0f}% reduction in MAE\n'.format(100 - qe_uv_error / qe_dft_error * 100)
121
122 print ''
123
124 print '#+CAPTION: All oxidation energies with GBRV PPs. E_{exp}, E_{DFT} and E_{DFT+\textit{U}(\V)} are the experimental, DFT and
125 print '#+ATTR_LATEX: :placement [H] :align c|c|c|c|c|c'
126 print '#+TBLNAME: all-GBRV'
127 print '|Reaction|E_{Exp}|E_{DFT}|E_{DFT+\textit{U}(\V)}|- E_{DFT+\textit{U}(\V)}|+ E_{DFT+\textit{U}(\V)}|'
128 print '|--|'
129
130 gbrv_dft_error, gbrv_uv_error = calc_error(0.487, atom_fits_gbrv, RXNS, gbrv_data, -874.942962698, 0.407, 0.567)
131
132 print ''
133
134 print 'GBRV: DFT MAE = {0:1.3f}, DFT+U(V) MAE = {1:1.3f}, x=0.487 +- 0.08'.format(gbrv_dft_error, gbrv_uv_error)
135 print '{0:1.0f}% reduction in MAE'.format(100 - gbrv_uv_error / gbrv_dft_error * 100)

```

TABLE XV. All oxidation energies with QE PPs. E_{exp} , E_{DFT} and $E_{\text{DFT}+U(\text{V})}$ are the experimental, DFT and DFT+ $U(\text{V})$ computed oxidation energies, respectively. $-E_{\text{DFT}+U(\text{V})}$ and $+E_{\text{DFT}+U(\text{V})}$ contain the 95% confidence interval of the oxidation energy calculated from the 95% confidence interval of x previously.

Reaction	E_{Exp}	E_{DFT}	$E_{\text{DFT}+U(\text{V})}$	$-E_{\text{DFT}+U(\text{V})}$	$+E_{\text{DFT}+U(\text{V})}$
$\text{VO} \rightarrow \text{V}_2\text{O}_5$	-3.558	-5.698	-3.555	0.190	0.190
$\text{VO} \rightarrow \text{VO}_2$	-2.923	-4.436	-2.652	0.158	0.158
$\text{VO} \rightarrow \text{V}_2\text{O}_3$	-1.871	-2.684	-1.600	0.096	0.096
$\text{V}_2\text{O}_3 \rightarrow \text{V}_2\text{O}_5$	-1.687	-3.014	-1.955	0.094	0.094
$\text{V}_2\text{O}_3 \rightarrow \text{VO}_2$	-1.052	-1.753	-1.052	0.062	0.062
$\text{VO}_2 \rightarrow \text{V}_2\text{O}_5$	-0.635	-1.261	-0.903	0.032	0.032
$\text{Cr}_2\text{O}_3 \rightarrow \text{CrO}_3$	-0.204	-2.596	-0.259	0.207	0.207
$\text{MnO} \rightarrow \text{MnO}_2$	-1.410	-2.871	-1.592	0.113	0.113
$\text{MnO} \rightarrow \text{Mn}_3\text{O}_4$	-0.804	-1.366	-1.059	0.027	0.027
$\text{Mn}_3\text{O}_4 \rightarrow \text{MnO}_2$	-0.606	-1.505	-0.533	0.086	0.086
$\text{FeO} \rightarrow \text{Fe}_2\text{O}_3$	-1.630	-1.796	-1.712	0.007	0.007
$\text{FeO} \rightarrow \text{Fe}_3\text{O}_4$	-1.240	-1.437	-1.221	0.019	0.019
$\text{Fe}_3\text{O}_4 \rightarrow \text{Fe}_2\text{O}_3$	-0.390	-0.359	-0.491	0.012	0.012
$\text{CoO} \rightarrow \text{Co}_3\text{O}_4$	-0.680	-1.224	-0.780	0.039	0.039

QE Library: DFT MAE = 0.955, DFT+ $U(\text{V})$ MAE = 0.139, $x=0.599 \pm 0.053$ 85% reduction in MAE

TABLE XVI. All oxidation energies with GBRV PPs. E_{exp} , E_{DFT} and $E_{\text{DFT}+U(V)}$ are the experimental, DFT and DFT+ $U(V)$ computed oxidation energies, respectively. $-E_{\text{DFT}+U(V)}$ and $+E_{\text{DFT}+U(V)}$ contain the 95% confidence interval of the oxidation energy calculated from the 95% confidence interval of x previously.

Reaction	E_{Exp}	E_{DFT}	$E_{\text{DFT}+U(V)}$	$-E_{\text{DFT}+U(V)}$	$+E_{\text{DFT}+U(V)}$
$\text{VO} \rightarrow \text{V}_2\text{O}_5$	-3.558	-5.666	-4.208	0.240	0.240
$\text{VO} \rightarrow \text{VO}_2$	-2.923	-4.420	-2.930	0.245	0.245
$\text{VO} \rightarrow \text{V}_2\text{O}_3$	-1.871	-2.609	-1.929	0.112	0.112
$\text{V}_2\text{O}_3 \rightarrow \text{V}_2\text{O}_5$	-1.687	-3.057	-2.278	0.128	0.128
$\text{V}_2\text{O}_3 \rightarrow \text{VO}_2$	-1.052	-1.811	-1.001	0.133	0.133
$\text{VO}_2 \rightarrow \text{V}_2\text{O}_5$	-0.635	-1.246	-1.278	0.005	0.005
$\text{Cr}_2\text{O}_3 \rightarrow \text{CrO}_3$	-0.204	-2.705	0.259	0.487	0.487
$\text{MnO} \rightarrow \text{MnO}_2$	-1.410	-2.923	-1.399	0.250	0.250
$\text{MnO} \rightarrow \text{Mn}_3\text{O}_4$	-0.804	-1.399	-0.829	0.094	0.094
$\text{Mn}_3\text{O}_4 \rightarrow \text{MnO}_2$	-0.606	-1.524	-0.570	0.157	0.157
$\text{FeO} \rightarrow \text{Fe}_2\text{O}_3$	-1.630	-1.852	-1.604	0.041	0.041
$\text{FeO} \rightarrow \text{Fe}_3\text{O}_4$	-1.240	-1.474	-1.135	0.056	0.056
$\text{Fe}_3\text{O}_4 \rightarrow \text{Fe}_2\text{O}_3$	-0.390	-0.378	-0.469	0.015	0.015
$\text{CoO} \rightarrow \text{Co}_3\text{O}_4$	-0.680	-1.379	-0.760	0.102	0.102

GBRV: DFT MAE = 0.984, DFT+ $U(V)$ MAE = 0.202, $x=0.487 \pm 0.08$ 79% reduction in MAE

D. Graph test/training set and DFT/DFT+ $U(V)$ parity plot

The following code takes the data in Tables XVI and XV and constructs Figure 5 in the manuscript. It also prints out all MAE values in a table.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib import rc, rcParams
4
5 # First load all of the data in lists

```

```

6 rxns, train_QE_exp, train_QE_uv = zip(*train_QE)
7 rxns, test_QE_exp, test_QE_uv = zip(*test_QE)
8 rxns, train_GBRV_exp, train_GBRV_uv = zip(*train_GBRV)
9 rxns, test_GBRV_exp, test_GBRV_uv = zip(*test_GBRV)
10 rxns, exp, QE_dft, QE_uv, QE_uv_low, QE_uv_high = zip(*all_QE)
11 rxns, exp, GBRV_dft, GBRV_uv, GBRV_uv_low, GBRV_uv_high = zip(*all_GBRV)
12
13 rcParams['mathtext.default'] = 'regular'
14
15 # First plot the training/test set reaction plot
16
17 fig = plt.figure(1, (3.25, 5.5))
18 ax = fig.add_axes([0.2, 0.55, 0.75, 0.42])
19 ax.plot((-6, 0.5), (-6, 0.5), ls='--', c='k')
20 ax.plot(train_QE_uv, train_QE_exp, ms=9,
21         marker='o', ls='none', c='gray', label='/')
22 ax.plot(test_QE_uv, test_QE_exp, ms=9,
23         marker='o', ls='none', c='k', label='/')
24 ax.plot(train_GBRV_uv, train_GBRV_exp, ms=9,
25         marker='^', ls='none', c='gray', label='Training set (QE/GBRV PPs)')
26 ax.plot(test_GBRV_uv, test_GBRV_exp, ms=9,
27         marker='^', ls='none', c='k', label='Test set with (QE/GBRV PPs)')
28
29 ax.set_xlim(-6, 2)
30 ax.set_xticks([-6, -4, -2, 0, 2])
31 ax.set_ylim(-5.5, 0.5)
32 ax.set_yticks([0, -1, -2, -3, -4, -5])
33 ax.set_ylabel(r'$\Delta H_{rxn}^{exp}$ (eV/M)')
34 ax.set_xticklabels([])
35 ax.text(-5.5, -0.1, '(a)')
36
37 ax.legend(loc=4, ncol=2, numpoints=1, columnspacing=0.0001, prop={'size':8},
38         handletextpad=0.1)
39
40 print '|Method|MAE (eV/M)|'
41 print '|--|'
42
43 # Now plot the DFT/DFT+U(V) with confidence intervals plot set reaction plot
44
45 QE_xerr_bars = np.vstack((QE_uv_low, QE_uv_high))
46 GBRV_xerr_bars = np.vstack((GBRV_uv_low, GBRV_uv_high))
47
48 ax = fig.add_axes([0.2, 0.11, 0.75, 0.42])
49
50 ax.plot((-6, 0.5), (-6, 0.5), ls='--', c='k')
51
52 # Plot DFT energies

```

```

53 dft_qe, = ax.plot(QE_dft, exp, marker='o', ls='none', c='r', label='/', ms=9)
54 dft_gbrv, = ax.plot(GBRV_dft, exp, marker='^', ls='none', c='r', label='DFT (QE/GBRV PPs)', ms=9)
55
56 dft_qe_mae = np.sum(np.absolute((np.array(QE_dft) - np.array(exp)))) / len(exp)
57 dft_gbrv_mae = np.sum(np.absolute((np.array(GBRV_dft) - np.array(exp)))) / len(exp)
58
59 print 'DFT (QE PPs)|{0:1.2f}|'.format(dft_qe_mae)
60 print 'DFT (GBRV PPs)|{0:1.2f}|'.format(dft_gbrv_mae)
61
62 # Plot energies computed using DFT+U and HSE06
63 E_dftu = zip(*ceder_data)[1]
64 E_hse06 = zip(*ceder_data)[2]
65
66 dftu_mae = np.sum(np.absolute((np.array(E_dftu) - np.array(exp)))) / len(exp)
67 hse06_mae = np.sum(np.absolute((np.array(E_hse06) - np.array(exp)))) / len(exp)
68
69 print 'DFT+U|{0:1.2f}|'.format(dftu_mae)
70 print 'HSE06|{0:1.2f}|'.format(hse06_mae)
71
72 dftu, = ax.plot(E_dftu, exp, marker='s', c='orange', ls='none', ms=9, label='/')
73 hse06, = ax.plot(E_hse06, exp, marker='s', c='yellow', ls='none', ms=9, label=r'DFT+ $\mathit{U}$ /HSE06')
74
75 dftuv_qe = ax.errorbar(QE_uv, exp, xerr=QE_xerrBars, marker='o', ls='none', ms=9,
76                        c='c', label=r'DFT+ $\mathit{U}$ (V)', elinewidth=2, capthick=2,
77                        capsize=5)
78 dftuv_gbrv = ax.errorbar(GBRV_uv, exp, xerr=GBRV_xerrBars, marker='^', ls='none', ms=9,
79                          c='c', label=r'DFT+ $\mathit{U}$ (V)', elinewidth=2, capthick=2,
80                          capsize=5)
81
82 dftuv_qe_mae = np.sum(np.absolute((np.array(QE_uv) - np.array(exp)))) / len(exp)
83 dftuv_gbrv_mae = np.sum(np.absolute((np.array(GBRV_uv) - np.array(exp)))) / len(exp)
84
85 print 'DFT+U(V) (QE PPs)|{0:1.2f}|'.format(dftuv_qe_mae)
86 print 'HSE06 (GBRV PPs)|{0:1.2f}|'.format(dftuv_gbrv_mae)
87
88 ax.set_xlim(-6, 2)
89 ax.set_xticks([-6, -4, -2, 0, 2])
90 ax.set_ylim(-5.5, 0.5)
91 ax.set_yticks([0, -1, -2, -3, -4, -5])
92 ax.set_xlabel(r' $\Delta H_{\text{rxn}}^{\text{calc}}$  (eV/M)')
93 ax.set_ylabel(r' $\Delta H_{\text{rxn}}^{\text{exp}}$  (eV/M)')
94 ax.text(-5.5, -0.1, '(b)')
95
96 ax.legend([dft_qe, dftuv_qe, dftu, dft_gbrv, dftuv_gbrv, hse06],
97           ['/', '/', '/', 'DFT (QE / GBRV PPs)', r'DFT+ $\mathit{U}$ (V) (QE / GBRV PPs)', r'DFT+ $\mathit{U}$  / HSE06'],
98           loc=4, ncol=2, numpoints=1, columnspacing=0.0001, prop={'size':8},
99           handletextpad=0.1)

```

```

100
101 plt.savefig('figures/FIG5.png', dpi=300)
102 plt.savefig('figures/FIG5.eps', dpi=300)
103
104 plt.show()

```

```

|Method|MAE (eV/M)|
|--|
|DFT (QE PPs)|0.96|
|DFT (GBRV PPs)|0.98|
|DFT+U|0.23|
|HSE06|0.40|
|DFT+U(V) (QE PPs)|0.14|
|HSE06 (GBRV PPs)|0.20|

```

Method	MAE (eV/M)
DFT (QE PPs)	0.96
DFT (GBRV PPs)	0.98
DFT+U	0.23
HSE06	0.40
DFT+U(V) (QE PPs)	0.14
DFT+U(V) (GBRV PPs)	0.20

VIII. OXIDATION ENERGIES COMPUTED WITH DFT+U AND HSE06

This section details the acquisition and storage of total energy of data of the compounds we tested using DFT+ U values using a constant U and the HSE05 hybrid functional. DFT+ U and HSE05 values were acquired from both three sources from the Ceder group [14–16].

A. DFT+ U total energies with element specific U values

The table below lists the total energies (eV/atom) done at an element specific U . The element specific U was typically chosen to minimize errors in oxidation energies. Vanadium

oxide values in addition to MnO_2 we're taken from materialsproject.org [14], while all other values we're taken from Hautier, G., et al. (2012) PRB, 85(155208) [15].

TABLE XVII. Total energies and Hubbard U values read from the materialsproject.org and Hautier, G., et al. (2012) PRB, 85(155208)

Compound	$E_{\text{tot}}^{\text{DFT}+U}$ (eV/atom)	U (eV)
VO	-7.7737	3.25
V_2O_3	-8.0208	3.25
VO_2	-7.7750	3.25
V_2O_5	-7.4420	3.25
Cr_2O_3	-7.888	3.5
CrO_3	-6.565	3.5
MnO	-7.8563	3.9
Mn_3O_4	-7.6875	3.9
MnO_2	-7.1175	3.9
FeO	-6.515	4.0
Fe_3O_4	-6.805	4.0
Fe_2O_3	-6.781	4.0
CoO	-5.62	3.4
Co_3O_4	-6.0802	3.4

B. HSE06 offset formation energies

The table below is a table of offset formation energies directly read off of Figure 3 in Chevrier, V., et al. (2010). PRB, 82(075122) [16]. All formation energies are referenced to -15 eV, so that the actual formation energy is $\Delta E_{\text{form}}^{\text{HSE06}} - 15$ eV.

TABLE XVIII. Formation energies read from Figure 3 in Chevrier, V., et al. (2010). PRB, 82(075122).

Compound	$\Delta E_{\text{form}}^{\text{HSE06}}$ (eV/O ₂)
VO	4.861
V ₂ O ₃	5.647
VO ₂	7.219
V ₂ O ₅	8.492
Cr ₂ O ₃	7.317
CrO ₃	12.134
MnO	6.188
Mn ₃ O ₄	7.338
MnO ₂	9.868
FeO	8.858
Fe ₃ O ₄	8.541
Fe ₂ O ₃	9.022
CoO	8.605
Co ₃ O ₄	9.573

C. Storing DFT+U and HSE06 energies with respect to metal atom (eV/M) in a single table

The code below stores the acquired DFT+*U* total energies (eV/atom) and HSE06 formation energies (eV/O₂) and converts them so they are both with respect to the number of metal atoms, which was the convention we have used in this manuscript. The table produced is then read by the next section in computing the oxidation energy reactions we used in this study.

```

1 import numpy as np
2 compounds, E_dftu = zip(*dftu_data)
3 compounds, E_hse06 = zip(*hse06)
4
5 stoic = {'VO':          {'O2conv': 1.0 / 2.0,          'atomsconv': 2.0          },
6         'V_{2}O_{3}': {'O2conv': (3.0 / 2.0) / 2.0, 'atomsconv': 5.0 / 2.0},

```

```

7      'VO_{2}':      {'02conv': 2.0 / 2.0,      'atomsconv': 3.0      },
8      'V_{2}O_{5}': {'02conv': (5.0 / 2.0) / 2.0, 'atomsconv': 7.0 / 2.0},
9      'Cr_{2}O_{3}': {'02conv': (3.0 / 2.0) / 2.0, 'atomsconv': 5.0 / 2.0},
10     'CrO_{3}':      {'02conv': 3.0 / 2.0,      'atomsconv': 4.0      },
11     'MnO':          {'02conv': 1.0 / 2.0,      'atomsconv': 2.0      },
12     'Mn_{3}O_{4}': {'02conv': (4.0 / 3.0) / 2.0, 'atomsconv': 7.0 / 3.0},
13     'MnO_{2}':      {'02conv': 2.0 / 2.0,      'atomsconv': 3.0      },
14     'FeO':          {'02conv': 1.0 / 2.0,      'atomsconv': 2.0      },
15     'Fe_{3}O_{4}': {'02conv': (4.0 / 3.0) / 2.0, 'atomsconv': 7.0 / 3.0},
16     'Fe_{2}O_{3}': {'02conv': (3.0 / 2.0) / 2.0, 'atomsconv': 5.0 / 2.0},
17     'CoO':          {'02conv': 1.0 / 2.0,      'atomsconv': 2.0      },
18     'Co_{3}O_{4}': {'02conv': (4.0 / 3.0) / 2.0, 'atomsconv': 7.0 / 3.0},
19 }
20
21 E_hse06 = (np.array(E_hse06) - 15)
22
23 print '|Compound|E_{DFT+U}|E_{HSE06}|'
24 print '|---|'
25
26 for compound, e_dftu, e_hse06 in zip(compounds, E_dftu, E_hse06):
27     e_dftu = e_dftu * stoic[compound]['atomsconv']
28     e_hse06 = e_hse06 * stoic[compound]['02conv']
29     print '|{compound}|{e_dftu:1.3f}|{e_hse06}|'.format(**locals())

```

TABLE XIX. DFT+ U total energies and HSE06 formation energies, now with respect to the number of metal ions in the compound (eV/M)

Compound	$E_{\text{DFT}+U}$ (eV/M)	E_{HSE06} (eV/M)
VO	-15.547	-5.0695
V2O3	-20.052	-7.01475
VO2	-23.325	-7.781
V2O5	-26.047	-8.135
Cr2O3	-19.720	-5.76225
CrO3	-26.260	-4.299
MnO	-15.713	-4.406
Mn3O4	-17.938	-5.108
MnO2	-21.352	-5.132
FeO	-13.030	-3.071
Fe3O4	-15.878	-4.306
Fe2O3	-16.953	-4.4835
CoO	-11.240	-3.1975
Co3O4	-14.187	-3.618

D. Calculate oxidation reaction energies

We now calculate the oxidation reaction energies with the code below. Because the DFT+ U energies are given as total energies, we also include the total energy of O_2 (-9.849707 eV) calculated in VASP with the same calculation parameters for the calculation of reaction energies in addition to the 1.36 eV correction.

```

1  O2 = -9.849707 + 1.36
2
3  RXNS = [('V2O5', 'VO', 0.75),      # r1: VO + 3/4 O2 = VO_2.5
4         ('VO2', 'VO', 0.5),         # r2: VO + 1/2 O2 = VO_2
5         ('V2O3', 'VO', 0.25),       # r3: VO + 1/4 O2 = VO_1.5
6         ('V2O5', 'V2O3', 0.5),      # r4: VO_1.5 + 1/2 O2 = VO_2.5
7         ('VO2', 'V2O3', 0.25),      # r5: VO_1.5 + 1/4 O2 = VO_2
8         ('V2O5', 'VO2', 0.25),      # r6: VO_2 + 1/4 O2 = VO_2.5
9         ('CrO3', 'Cr2O3', 0.75),    # r7: CrO_1.5 + 3/4 O2 = CrO_3

```

```

10         ('MnO2', 'MnO', 0.5),          # r8: MnO + 1/2 O2 = MnO_2
11         ('Mn3O4', 'MnO', 1.0/6.0),     # r9: MnO + 1/6 O2 = MnO_1.33
12         ('MnO2', 'Mn3O4', 1.0/3.0),     # r10: MnO_1.33 + 1/3 O2 = MnO_2
13         ('Fe2O3', 'FeO', 0.25),        # r11: FeO + 1/4 O2 = FeO_1.5
14         ('Fe3O4', 'FeO', 1.0/6.0),     # r12: FeO + 1/6 O2 = FeO_1.33
15         ('Fe2O3', 'Fe3O4', 1.0/12.0),  # r13: FeO_1.33 + 1/12 O2 = FeO_1.5
16         ('Co3O4', 'CoO', 1.0/6.0),     # r14: CoO + 1/6 O2 = CoO_1.33
17     ]
18
19     E = {}
20
21     print '|Reaction|E_{DFT+U}|E_{HSE06}|'
22     print '|---|'
23
24     for name, E_dftu, E_hse06 in data:
25         E[name] = {'E_dftu': E_dftu, 'E_hse06': E_hse06}
26
27     for prod, react, stoic in RXNS:
28         rxn_dftu = E[prod]['E_dftu'] - E[react]['E_dftu'] - stoic*02
29         rxn_hse06 = E[prod]['E_hse06'] - E[react]['E_hse06']
30         rxn = ''
31         for letter in react:
32             if letter.isdigit():
33                 rxn += '_{' + letter + '}'
34             else:
35                 rxn += letter
36         rxn += ' $\rightarrow$ '
37         for letter in prod:
38             if letter.isdigit():
39                 rxn += '_{' + letter + '}'
40             else:
41                 rxn += letter
42         print '|{rxn}|{rxn_dftu}|{rxn_hse06}|'.format(**locals())

```

TABLE XX. DFT+ U total energies and HSE06 oxidation energies.

Reaction	$E_{\text{DFT+U}}$	E_{HSE06}
$\text{VO} \rightarrow \text{V}_2\text{O}_5$	-4.13271975	-3.0655
$\text{VO} \rightarrow \text{VO}_2$	-3.5331465	-2.7115
$\text{VO} \rightarrow \text{V}_2\text{O}_3$	-2.38257325	-1.94525
$\text{V}_2\text{O}_3 \rightarrow \text{V}_2\text{O}_5$	-1.7501465	-1.12025
$\text{V}_2\text{O}_3 \rightarrow \text{VO}_2$	-1.15057325	-0.76625
$\text{VO}_2 \rightarrow \text{V}_2\text{O}_5$	-0.59957325	-0.354
$\text{Cr}_2\text{O}_3 \rightarrow \text{CrO}_3$	-0.17271975	1.46325
$\text{MnO} \rightarrow \text{MnO}_2$	-1.3941465	-0.726
$\text{MnO} \rightarrow \text{Mn}_3\text{O}_4$	-0.810048833333	-0.702
$\text{Mn}_3\text{O}_4 \rightarrow \text{MnO}_2$	-0.584097666667	-0.024
$\text{FeO} \rightarrow \text{Fe}_2\text{O}_3$	-1.80057325	-1.4125
$\text{FeO} \rightarrow \text{Fe}_3\text{O}_4$	-1.43304883333	-1.235
$\text{Fe}_3\text{O}_4 \rightarrow \text{Fe}_2\text{O}_3$	-0.367524416667	-0.1775
$\text{CoO} \rightarrow \text{Co}_3\text{O}_4$	-1.53204883333	-0.4205

IX. DFT+ U (V) FORMATION ENERGIES OF OXIDES

In addition to computing oxidation energies of oxides, we also computed formation energies, which are their oxidation energies with respect to the transition metal in its metallic form. Though we found improvement in some materials, we did not find the same universal improvement in MAE as we did in the oxidation energies for both sets of pseudopotentials. The results are below for each set of pseudopotential.

A. QE library PPs

The following code goes through the formation energy of each oxide we looked at with the QE library of PPs and graphs the error at different values of x . Though we find that in some cases an x value near the fitted parameters we found for oxidation energies (0.4-0.6), a large majority of cases the experimental value is never reached for any x value.

```

1  import sys
2  import numpy as np
3  from scipy.stats.distributions import t
4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)
37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41 for element in elements:
42     atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                         atom_data_gbrv[element]['E'], kind='cubic')
44
45 ## Now calculate the oxidation energies using each pseudopotential
46 # Create a list of indexes that match the differences between species. The first

```

```

47  # index will be the oxidized product
48  RXNS = [
49      ('V2O5', 'V'),      # r1: V + 5/4 O2 = VO_2.5
50      ('VO2', 'V'),      # r2: V + O2 = VO_2
51      ('V2O3', 'V'),     # r3: V + 3/4 O2 = VO_1.5
52      ('VO', 'V'),       # r4: V + 1/2 O2 = VO
53      ('Fe2O3', 'Fe'),   # r8: Fe + 3/4 O2 = FeO_1.5
54      ('Fe3O4', 'Fe'),   # r9: Fe + 2/4 O2 = FeO_1.333
55      ('FeO', 'Fe'),     # r10: Fe + 1/2 O2 = FeO
56      ('CoO', 'Co'),     # r11: Co + 1/2 O2 = CoO
57      ('Co3O4', 'Co'),   # r12: Co + 2/4 O2 = CoO_1.333
58      ('NiO', 'Ni'),     # r13: Ni + 1/2 O2 = NiO
59  ]
60
61  def calc_error(x, atom_data, pair, bulk_data, O2):
62      bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
63      O2 += 1.27580582171
64      ys, Es = [], []
65      Ubcohs, UOcohs = {}, {}
66      cohavgs, Ofrac, tot_energy, enthalpies = {}, {}, {}, {}
67      for atoms, EO, EUcalc, Ucalc, O_frac, exp in bulk_data:
68          element = atoms.translate(None, '00123456789')
69          bulkUcoh = EUcalc - atom_data[element](Ucalc) - O_frac * O2 / 2
70          atomUcoh = EO - atom_data[element](0) - O_frac * O2 / 2
71          avgUcoh = x * bulkUcoh + (1 - x) * atomUcoh
72          cohavgs[atoms] = avgUcoh
73          Ofrac[atoms] = O_frac
74          enthalpies[atoms] = exp
75
76      oxi = cohavgs[pair[0]] - cohavgs[pair[1]]
77      exp_oxi = enthalpies[pair[0]] - enthalpies[pair[1]]
78
79      return oxi - exp_oxi
80
81  for rxn in RXNS:
82      errors = []
83      for i in np.linspace(0, 1, 11):
84          uv_error = calc_error(i, atom_fits_qe, rxn, qe_data, -864.185)
85          errors.append(uv_error)
86
87      plt.figure(1, (2.5, 3))
88      plt.xlabel(r'$x$')
89      plt.ylabel('Error (eV)')
90      plt.title('{0}-to-{1}'.format(rxn[1], rxn[0]))
91      plt.plot(np.linspace(0, 1, 11), errors, marker='o')
92      plt.axhline(0, ls='--', c='k')
93      plt.tight_layout()

```



```

94 plt.savefig('supporting-figures/x-vs-error-{0}-to-{1}-QE.png'.format(rxn[1], rxn[0]))
95 plt.close()
96
97 print '#+CAPTION: $x$ vs error of formation energy of \\ce{{{0}}}' with QE library PPs.'.format(rxn[0])
98 print '#+ATTR_LATEX: :placement [H] :placement [H] :width 2.5in'
99 print '[./supporting-figures/x-vs-error-{0}-to-{1}-QE.png]'.format(rxn[1], rxn[0])
100 print ''

```

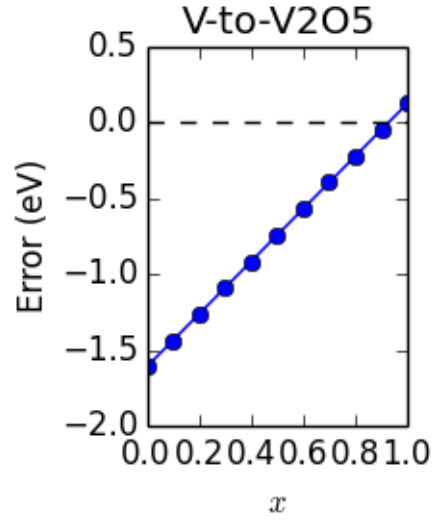


FIG. 2. x vs error of formation energy of V_2O_5 with QE library PPs.

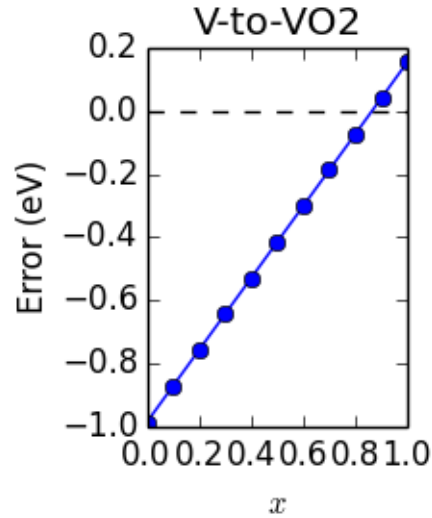


FIG. 3. x vs error of formation energy of VO_2 with QE library PPs.

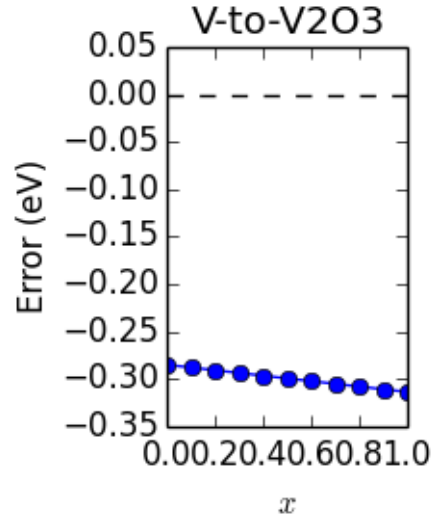


FIG. 4. x vs error of formation energy of V₂O₃ with QE library PPs.

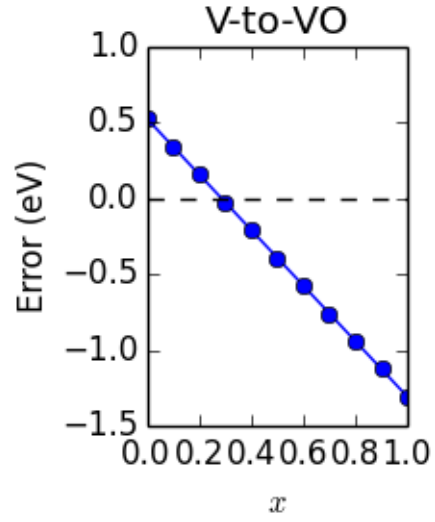


FIG. 5. x vs error of formation energy of VO with QE library PPs.

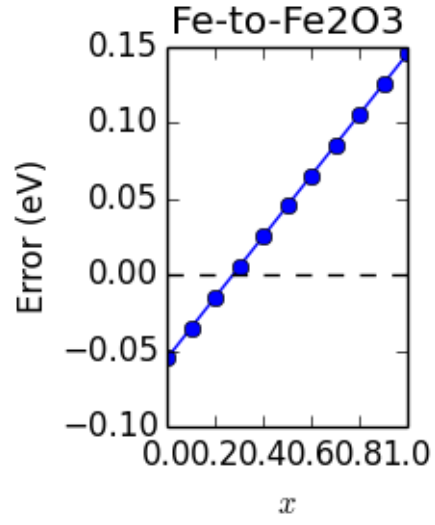


FIG. 6. x vs error of formation energy of Fe_2O_3 with QE library PPs.

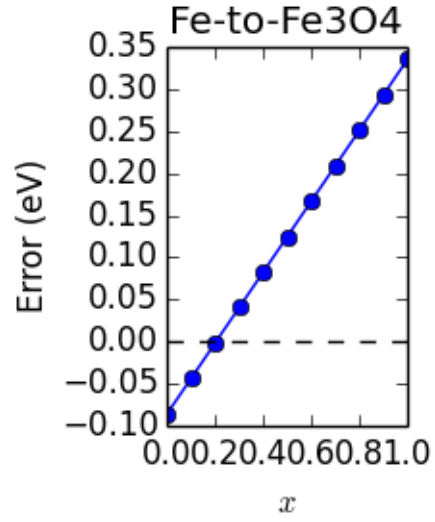


FIG. 7. x vs error of formation energy of Fe_3O_4 with QE library PPs.

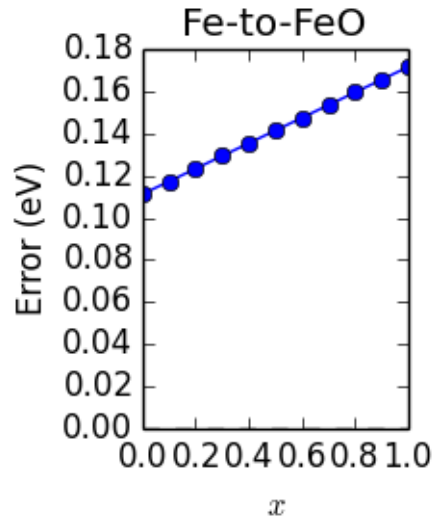


FIG. 8. x vs error of formation energy of FeO with QE library PPs.

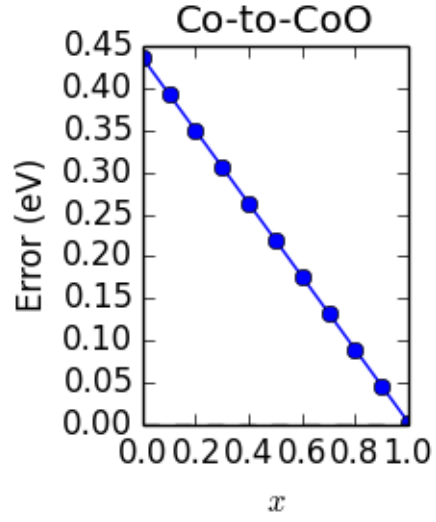


FIG. 9. x vs error of formation energy of CoO with QE library PPs.

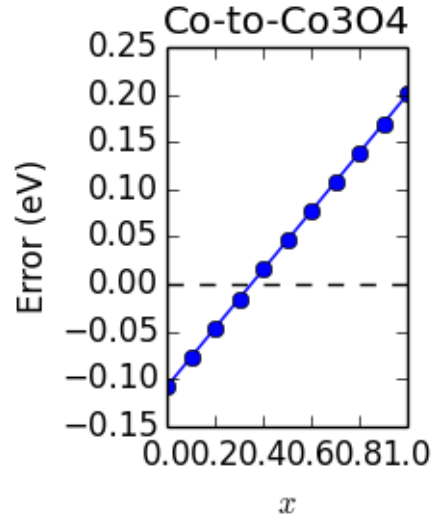


FIG. 10. x vs error of formation energy of Co_3O_4 with QE library PPs.

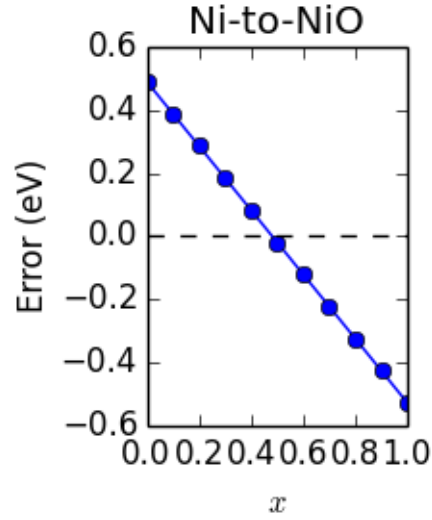


FIG. 11. x vs error of formation energy of NiO with QE library PPs.

B. GBRV PPs

The following code goes through the formation energy of each oxide we looked at with the GBRV library of PPs and graphs the error at different values of x . Though we find that in some cases an x value near the fitted parameters we found for oxidation energies (0.4

0.6), a large majority of cases the experimental value is never reached for any x value.

```
1  import sys
2  import numpy as np
3  from scipy.stats.distributions import t
4  from pycse import nlinfit
5  from scipy.interpolate import interp1d
6  import matplotlib.pyplot as plt
7
8  ## First store the atom data from each PP in separate dictionaries
9
10 # Original QE Library data
11 special_atoms = ('Ni', 'Fe', 'V', 'Co', 'Mn')
12 atom_fits_qe = {}
13
14 for atom in special_atoms:
15     Us, Es = zip(*eval(atom + '_atom'))
16     atom_fits_qe[atom] = interp1d(Us[1:], Es[1:], kind='cubic')
17
18 # GBRV data
19 atom_data_gbrv, atom_fits_gbrv = {}, {}
20
21 elements = gbrv_atoms[0][1:]
22
23 # First collect the atom names
24
25 for atom in elements:
26     atom_data_gbrv[atom] = {}
27     atom_data_gbrv[atom]['U'] = []
28     atom_data_gbrv[atom]['E'] = []
29
30 # We now need some code to store the EvsU data in the atom_data dictionary. We
31 # first store the EvsU data in the atom_data dictionary
32
33 Us = zip(*gbrv_atoms)[0]
34 for energies in zip(*gbrv_atoms)[1:]:
35     for atom_U, energy in zip(Us[1:], energies[1:]):
36         # print type(energy)
37         if type(energy) == float:
38             atom_data_gbrv[energies[0]]['U'].append(atom_U)
39             atom_data_gbrv[energies[0]]['E'].append(energy)
40
41 for element in elements:
42     atom_fits_gbrv[element] = interp1d(atom_data_gbrv[element]['U'],
43                                         atom_data_gbrv[element]['E'], kind='cubic')
```

```

45  ## Now calculate the oxidation energies using each pseudopotential
46  # Create a list of indexes that match the differences between species. The first
47  # index will be the oxidized product
48  RXNS = [
49      ('V2O5', 'V'),      # r1: V + 5/4 O2 = VO_2.5
50      ('VO2', 'V'),       # r2: V + O2 = VO_2
51      ('V2O3', 'V'),      # r3: V + 3/4 O2 = VO_1.5
52      ('VO', 'V'),        # r4: V + 1/2 O2 = VO
53      ('Fe2O3', 'Fe'),     # r8: Fe + 3/4 O2 = FeO_1.5
54      ('Fe3O4', 'Fe'),     # r9: Fe + 2/4 O2 = FeO_1.333
55      ('FeO', 'Fe'),       # r10: Fe + 1/2 O2 = FeO
56      ('CoO', 'Co'),       # r11: Co + 1/2 O2 = CoO
57      ('Co3O4', 'Co'),     # r12: Co + 2/4 O2 = CoO_1.333
58      ('NiO', 'Ni'),       # r13: Ni + 1/2 O2 = NiO
59  ]
60
61  def calc_error(x, atom_data, pair, bulk_data, O2):
62      bulk_data = bulk_data[1:] # Ignore the first entry, which are the table headers
63      O2 += 1.27580582171
64      ys, Es = [], []
65      Ubcohs, UOcohs = {}, {}
66      cohavgs, Ofrac, tot_energy, enthalpies = {}, {}, {}, {}
67      for atoms, EO, EUcalc, Ucalc, O_frac, exp in bulk_data:
68          element = atoms.translate(None, '00123456789')
69          bulkUcoh = EUcalc - atom_data[element](Ucalc) - O_frac * O2 / 2
70          atomUcoh = EO - atom_data[element](O) - O_frac * O2 / 2
71          avgUcoh = x * bulkUcoh + (1 - x) * atomUcoh
72          cohavgs[atoms] = avgUcoh
73          Ofrac[atoms] = O_frac
74          enthalpies[atoms] = exp
75
76      oxi = cohavgs[pair[0]] - cohavgs[pair[1]]
77      exp_oxi = enthalpies[pair[0]] - enthalpies[pair[1]]
78
79      return oxi - exp_oxi
80
81  for rxn in RXNS:
82      errors = []
83      for i in np.linspace(0, 1, 11):
84          uv_error = calc_error(i, atom_fits_gbrv, rxn, gbrv_data, -874.942962698)
85          errors.append(uv_error)
86
87      plt.figure(1, (2.5, 3))
88      plt.xlabel(r'$x$')
89      plt.ylabel('Error (eV)')
90      plt.title('{0}-to-{1}'.format(rxn[1], rxn[0]))
91      plt.plot(np.linspace(0, 1, 11), errors, marker='o')

```

```

92     plt.axhline(0, ls='--', c='k')
93     plt.tight_layout()
94     plt.savefig('supporting-figures/x-vs-error-{0}-to-{1}-GBRV.png'.format(rxn[1], rxn[0]))
95     plt.close()
96
97     print '#+CAPTION: $x$ vs error of formation energy of \\ce{{{0}}}' with GBRV PPs.'.format(rxn[0])
98     print '#+ATTR_LATEX: :placement [H] :width 2.5in'
99     print '[./supporting-figures/x-vs-error-{0}-to-{1}-GBRV.png]'.format(rxn[1], rxn[0])
100    print ''

```

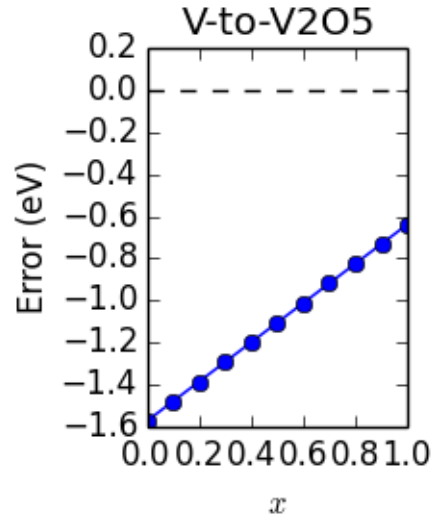


FIG. 12. x vs error of formation energy of V_2O_5 with GBRV PPs.

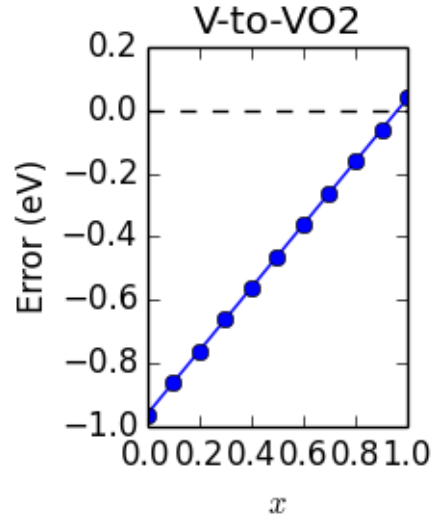


FIG. 13. x vs error of formation energy of VO_2 with GBRV PPs.

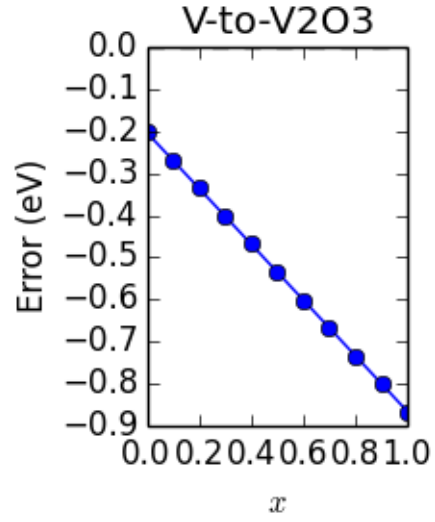


FIG. 14. x vs error of formation energy of V_2O_3 with GBRV PPs.

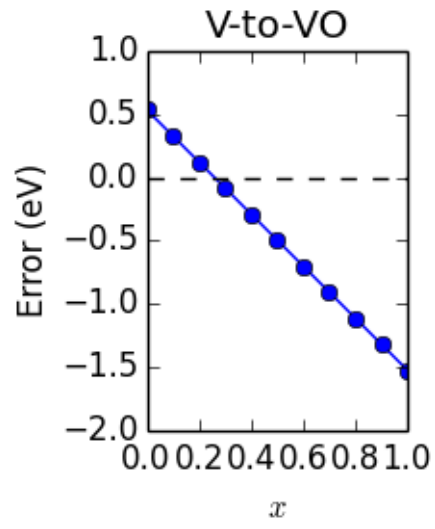


FIG. 15. x vs error of formation energy of VO with GBRV PPs.

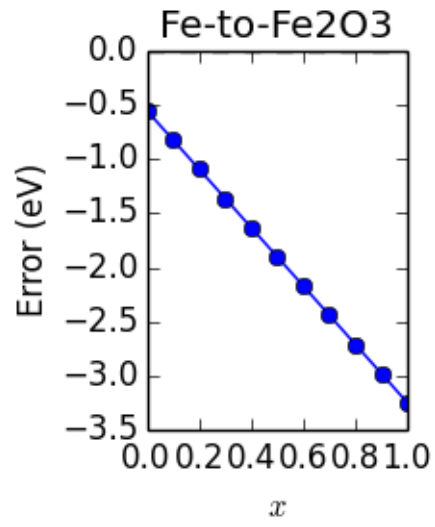


FIG. 16. x vs error of formation energy of Fe_2O_3 with GBRV PPs.

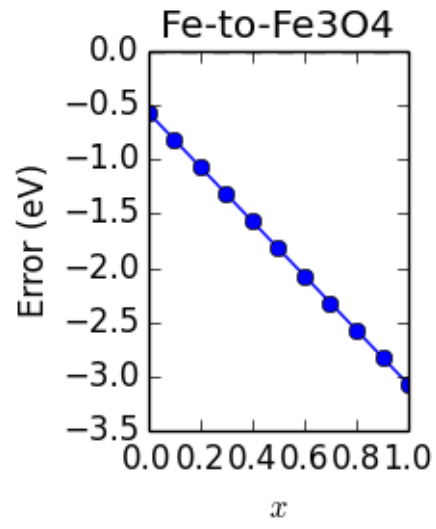


FIG. 17. x vs error of formation energy of Fe_3O_4 with GBRV PPs.

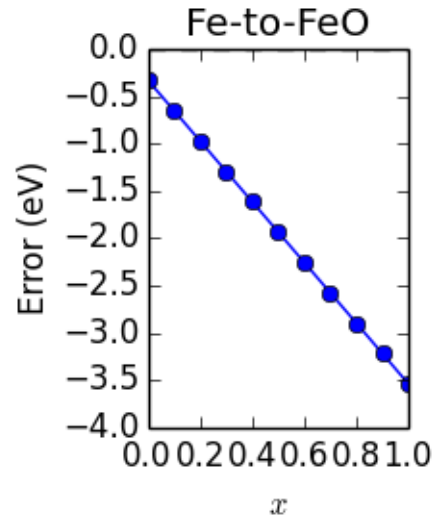


FIG. 18. x vs error of formation energy of FeO with GBRV PPs.

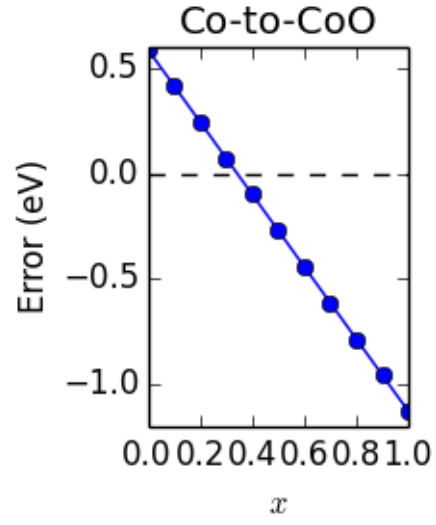


FIG. 19. x vs error of formation energy of CoO with GBRV PPs.

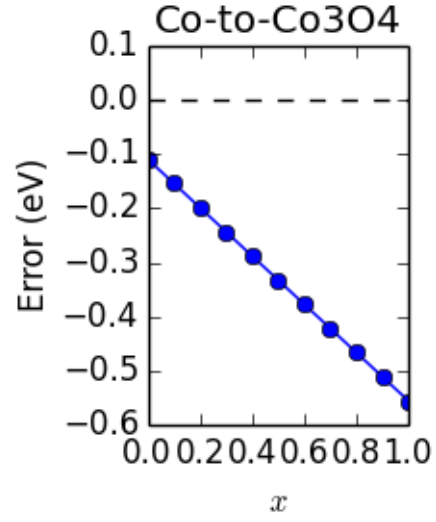


FIG. 20. x vs error of formation energy of Co_3O_4 with GBRV PPs.

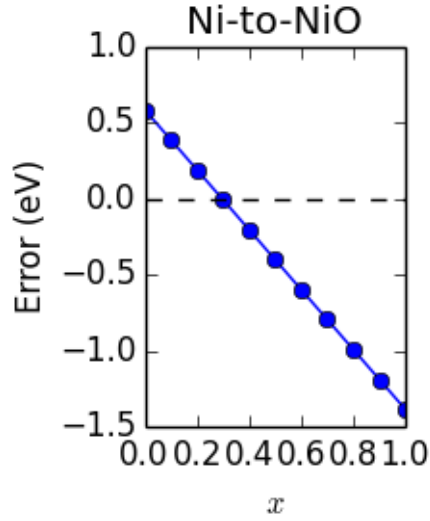


FIG. 21. x vs error of formation energy of NiO with GBRV PPs.

X. FINDING EQUILIBRIUM BULK STRUCTURES

Below is the code for calculating all of the equilibrium structures used in this paper. Note that all results below are done with the original QUANTUM-ESPRESSO PPs. Results for the GBRV PPs can be done with the same code but referencing different PPs by changing the espresso_PPs.py file (see espresso module in the **Required modules** section. The same calculation parameters were used to evaluate dependence of energy on the Hubbard U . For metals, we assumed the experimental crystal lattice, which was taken out of Kittel [17]. For oxides, please see the **Required modules** section at the end of the supporting information for the specifications of the crystal structures as well as the functions that produced the crystal structures. For bulk Mn, we assumed the bcc structure and found the lattice constant through performing a complete cell relaxation.

TABLE XXI. Crystal lattice coordinates of metal systems.

Element	Structure	a	c
V	bcc	3.03	
Fe	bcc	2.87	
Co	hcp	2.51	4.07
Ni	fcc	3.52	

All equilibrium volumes were calculated by constructing equations of states using the EquationOfState module found in the ase.utils.eos module. The equation of state used was a simple third order inverse polynomial fit [18]. All python code in the below sub-sections perform two functions of submitting calculations at multiple volumes and constructing the equation of state from finished calculations.

A. Vanadium oxides

1. VO

```

1  from espresso import *
2  from ase_addons.bulk import rocksalt
3
4  lats = (3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7)
5  energies, vols = [], []
6  ready = True
7
8  for lat in lats:
9      bulk = rocksalt(('V', 'O'), a=lat, mags=(0.5, 0))
10     with Espresso('VO-QE/EOS/1-{0:1.3f}'.format(lat), atoms=bulk,
11                  calculation='vc-relax',
12                  disk_io='none',
13                  ecutwfc=60.0, ecutrho=600.0,
14                  occupations='smearing', smearing='mp', degauss=0.01,
15                  nspin=2,
16                  mixing_beta=0.3,
17                  conv_thr=1e-8,
18                  cell_dofree='shape',
19                  kpts=(6, 6, 6),
20                  walltime='24:00:00') as calc:
21         try:
22             energies.append(bulk.get_potential_energy())

```

```

23         vols.append(bulk.get_volume())
24     except (EspressoSubmitted, EspressoRunning):
25         print calc.espressodir, 'running'
26         ready = False
27
28 if ready == False:
29     import sys; sys.exit()
30
31 from ase_addons.utils import *
32 eos = EquationOfState(vols, energies)
33 eos.plot('supporting-figures/VO-EOS-coarse-QE.png', show=True, title='VO-coarse')

```

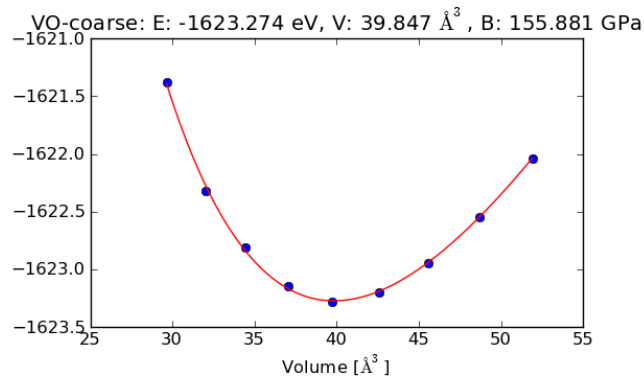


FIG. 22. VO Coarse EOS.

2. V_2O_3

```

1  from espresso import *
2  from ase_addons.bulk import corundum
3
4  ready = True
5
6  V2O3 = corundum(('V', 'O'), vol=100.3, mags=(0.7, 0))
7  cell10 = V2O3.get_cell()
8  v0 = V2O3.get_volume()
9  factors = (0.8, 0.9, 1.0, 1.1, 1.2)
10 energies, vols = [], []
11
12 for factor in factors:
13     bulk = V2O3.copy()
14     cell_factor = factor ** (1./3.)
15     bulk.set_cell(cell10 * cell_factor, scale_atoms=True)

```

```

16     with Espresso('V2O3-QE/EOS/1-{:0:1.3f}'.format(factor), atoms=bulk,
17                   disk_io='none', calculation='vc-relax',
18                   ecutwfc=60.0, ecutrho=600.0,
19                   occupations='smearing', smearing='mp', degauss=0.01,
20                   nspin=2,
21                   mixing_beta=0.3, conv_thr=1e-8,
22                   kpts=(6, 6, 6),
23                   cell_dofree='shape') as calc:
24         try:
25             energies.append(bulk.get_potential_energy())
26             vols.append(bulk.get_volume())
27         except (EspressoSubmitted, EspressoRunning):
28             print calc.espressodir, 'running'
29             ready = False
30
31 if ready == False:
32     import sys; sys.exit()
33
34 from ase_addons.utils import *
35 eos = EquationOfState(vols, energies)
36 eos.plot('supporting-figures/V2O3-EOS-coarse-QE.png', show=True, title='V2O3-coarse')

```

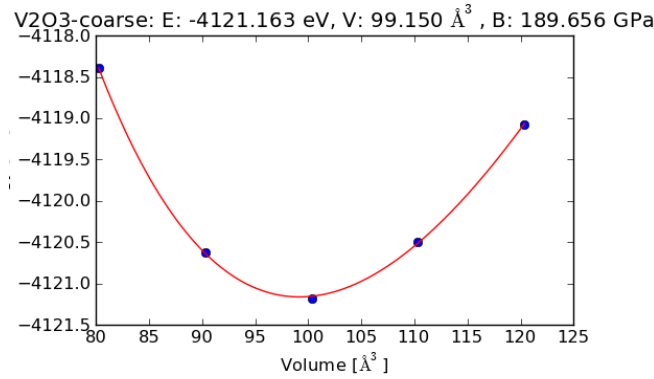


FIG. 23. V_2O_3 EOS in QE.

3. VO_2

```

1 from espresso import *
2 from ase_addons.bulk import rutile
3
4 ready = True

```



```

5
6  V02 = rutile(('V', 'O'), mags=(0.5,0))
7  cell10 = V02.get_cell()
8  v0 = V02.get_volume()
9  factors = (0.8, 0.9, 1.0, 1.1, 1.2)
10 energies, vols = [], []
11
12 for factor in factors:
13     bulk = V02.copy()
14     cell_factor = factor ** (1./3.)
15     bulk.set_cell(cell10 * cell_factor, scale_atoms=True)
16     with Espresso('V02-QE/EOS-magnetic/V-{0:1.3f}'.format(factor), atoms=bulk,
17                   disk_io='none', calculation='vc-relax',
18                   ecutwfc=60.0, ecutrho=600.0,
19                   occupations='smearing', smearing='mp', degauss=0.01,
20                   nspin=2,
21                   mixing_beta=0.3, conv_thr=1e-8,
22                   kpts=(7, 7, 7),
23                   cell_dofree='shape') as calc:
24
25         try:
26             energies.append(bulk.get_potential_energy())
27             vols.append(bulk.get_volume())
28         except (EspressoSubmitted, EspressoRunning):
29             print calc.espressodir, 'running'
30             ready = False
31
32 if ready == False:
33     import sys; sys.exit()
34
35 from ase_addons.utils import *
36 eos = EquationOfState(vols, energies)
37 eos.plot('supporting-figures/V02-EOS-coarse-QE.png', show=True, title='V02-coarse')

```

4. V_2O_5

```

1  from espresso import *
2  from ase_addons import A2B5
3
4  ready = True
5
6  V205 = A2B5(('V', 'O'), mags=(0.5, 0))
7  cell10 = V205.get_cell()
8  v0 = V205.get_volume()

```

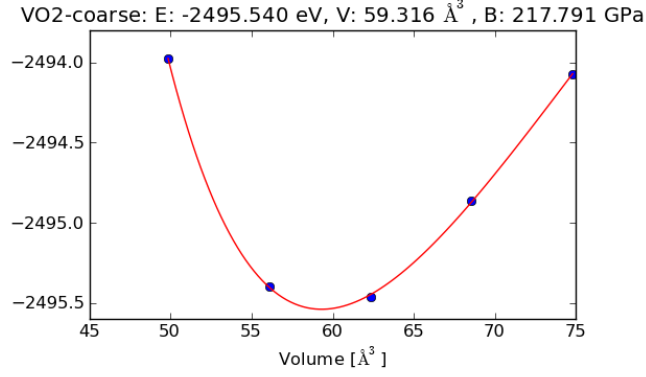


FIG. 24. VO₂ Coarse EOS in QE.

```

9  factors = (0.8, 0.9, 1.0, 1.1, 1.2)
10 energies, vols = [], []
11
12 for factor in factors:
13     bulk = V205.copy()
14     cell_factor = factor ** (1./3.)
15     bulk.set_cell(cell0 * cell_factor, scale_atoms=True)
16     with Espresso('V205-QE/EOS/V-{0:1.3f}'.format(factor), atoms=bulk,
17                  disk_io='none', calculation='vc-relax',
18                  ecutwfc=60.0, ecutrho=600.0,
19                  occupations='smearing', smearing='mp', degauss=0.01,
20                  nspin=2,
21                  mixing_beta=0.3, conv_thr=1e-8,
22                  kpts=(7, 7, 7),
23                  cell_dofree='shape') as calc:
24         try:
25             energies.append(bulk.get_potential_energy())
26             vols.append(bulk.get_volume())
27         except (EspressoSubmitted, EspressoRunning):
28             print calc.espressodir, 'running'
29             ready = False
30
31 if ready == False:
32     import sys; sys.exit()
33
34 from ase_addons.utils import *
35 eos = EquationOfState(vols, energies)
36 eos.plot('supporting-figures/V205-EOS-coarse-QE.png', show=True, title='V205-coarse')

```

The V₂O₅ structure is an oxide with alternating V₂O₅ layers held together by van der Waals forces. This explains the low curvature at large volumes. During the constant volume,

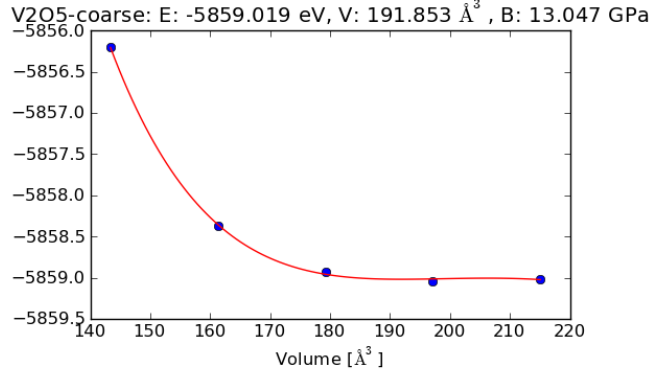


FIG. 25. V_2O_5 Coarse EOS in QE.

variable cell, shape relaxation at high volumes, the structure elongates in the direction perpendicular to the V_2O_5 planes, thereby lessening the lateral strain on the V_2O_5 layers. Because the layers are held together by van der Waals forces, elongation in the direction parallel to these forces leads to little changes in the overall energy. Similar EOS curves are seen for graphite. We therefore took the ground state volume to be the experimentally measured volume ($\approx 180 \text{ Å}^3$), considering it is close to the minimum energy achieved in our DFT calculations.

B. Manganese oxides

1. MnO

```

1 from espresso import *
2 from ase_addons import rocksalt
3
4 lats = (4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7)
5 energies, vols = [], []
6 ready = True
7
8 for lat in lats:
9     bulk = rocksalt(('Mn', 'O'), a=lat, mags=(0.7, 0))
10    with Espresso('MnO-QE/EOS/1-{0:1.3f}'.format(lat), atoms=bulk,
11                  calculation='vc-relax',
12                  disk_io='none',
13                  ecutwfc=60.0, ecutrho=600.0,
14                  occupations='smearing', smearing='mp', degauss=0.01,
```

```

15         nspin=2,
16         mixing_beta=0.3,
17         conv_thr=1e-8,
18         cell_dofree='shape',
19         kpts=(6, 6, 6),
20         walltime='24:00:00') as calc:
21     try:
22         energies.append(bulk.get_potential_energy())
23         vols.append(bulk.get_volume())
24     except (EspressoSubmitted, EspressoRunning):
25         print calc.espressodir, 'running'
26         ready = False
27
28 if ready == False:
29     import sys; sys.exit()
30
31 from ase_addons.utils import *
32 eos = EquationOfState(vols, energies)
33 eos.plot('supporting-figures/MnO-EOS-coarse-QE.png', show=True, title='MnO-coarse')

```

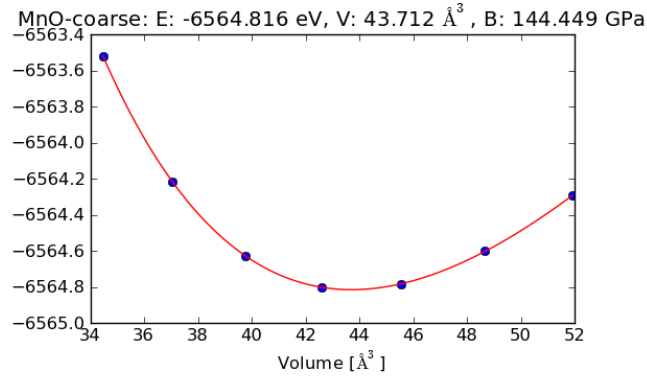


FIG. 26. EOS for MnO.

2. Mn_3O_4

```

1 from espresso import *
2 from ase_addons import Mn304
3
4 ready = True
5
6 bulk = Mn304(mags=[0.7, 0.0])
7 cell10 = bulk.get_cell()

```

```

8  v0 = bulk.get_volume()
9  # factors = (0.8, 0.9, 1.0, 1.1, 1.2)
10 factors = (0.8, 1.0, 1.1, 1.2)
11 energies, vols = [], []
12
13 for factor in factors:
14     atoms = bulk.copy()
15     cell_factor = factor ** (1./3.)
16     atoms.set_cell(cell0 * cell_factor, scale_atoms=True)
17     with Espresso('Mn304-QE/EOS/v-{factor:1.3f}'.format(**locals()), atoms=atoms,
18                  calculation='vc-relax',
19                  disk_io='none',
20                  ecutwfc=60.0, ecutrho=600.0,
21                  occupations='smearing', smearing='mp', degauss=0.01,
22                  nspin=2,
23                  mixing_beta=0.3,
24                  conv_thr=1e-8,
25                  cell_dofree='shape',
26                  kpts=(6, 6, 6),
27                  mem='16GB', ppn=4) as calc:
28         try:
29             energies.append(atoms.get_potential_energy())
30             vols.append(atoms.get_volume())
31         except (EspressoSubmitted, EspressoRunning):
32             print calc.espressodir, 'running'
33             ready = False
34
35 from ase_addons.utils import *
36 eos = EquationOfState(vols, energies)
37 eos.plot('supporting-figures/Mn304-EOS-coarse-QE.png', show=True, title='Mn304-coarse')

```

```

1  from espresso import *
2
3  factors = (0.8, 0.9, 1.0, 1.1, 1.2)
4  energies, vols = [], []
5
6  for factor in factors:
7      with Espresso('Mn304-QE/EOS/v-{factor:1.3f}'.format(**locals())) as calc:
8          energies.append(calc.get_potential_energy())
9          vols.append(calc.atoms.get_volume())
10
11 from ase_addons.utils import *
12 eos = EquationOfState(vols, energies)
13 eos.plot('supporting-figures/Mn304-EOS-coarse-QE.png', show=True, title='Mn304-coarse')

```

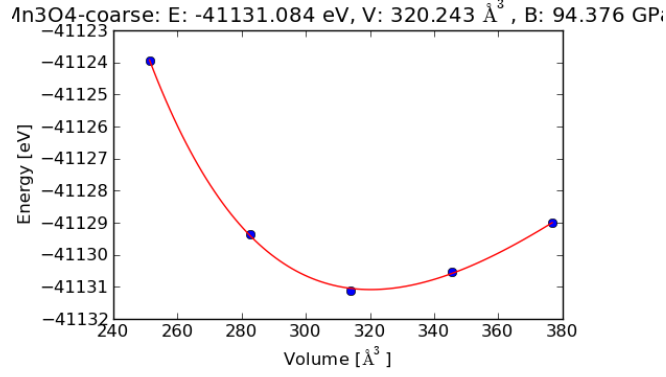


FIG. 27. Mn_3O_4 EOS using Quantum Espresso.

3. MnO_2

```

1  from espresso import *
2  from ase_addons import rutile
3
4  ready = True
5
6  MnO2 = rutile(('Mn', 'O'), mags=(0.5,0), a=4.3983, c=2.8730, u=0.30515)
7  cell0 = MnO2.get_cell()
8  v0 = MnO2.get_volume()
9  factors = (0.8, 0.9, 1.0, 1.1, 1.2)
10 energies, vols = [], []
11
12 for factor in factors:
13     bulk = MnO2.copy()
14     cell_factor = factor ** (1./3.)
15     bulk.set_cell(cell0 * cell_factor, scale_atoms=True)
16     with Espresso('MnO2-QE/EOS-magnetic/Mn-{0:1.3f}'.format(factor), atoms=bulk,
17                  disk_io='none', calculation='vc-relax',
18                  ecutwfc=60.0, ecutrho=600.0,
19                  occupations='smearing', smearing='mp', degauss=0.01,
20                  nspin=2,
21                  mixing_beta=0.3, conv_thr=1e-8,
22                  kpts=(7, 7, 7),
23                  cell_dofree='shape', walltime='24:00:00') as calc:
24
25         try:
26             energies.append(bulk.get_potential_energy())
27             vols.append(bulk.get_volume())
28         except (EspressoSubmitted, EspressoRunning):
29             print calc.espressodir, 'running'

```

```

30         ready = False
31
32     if ready == False:
33         import sys; sys.exit()
34
35     from ase_addons.utils import *
36     eos = EquationOfState(vols, energies)
37     eos.plot('supporting-figures/MnO2-EOS-coarse-QE.png', show=True, title='MnO2-coarse')

```

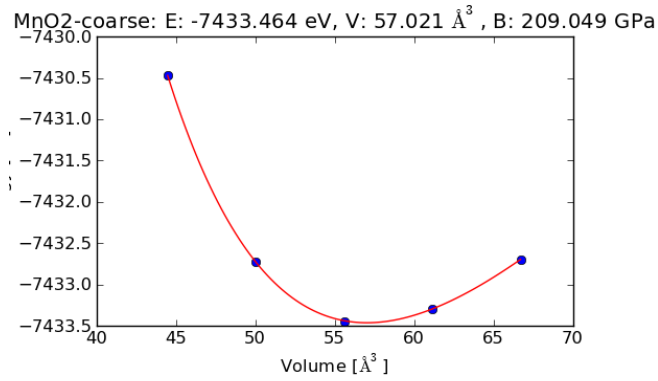


FIG. 28. MnO₂ Coarse EOS.

C. Iron oxides

1. FeO

```

1  from ase import Atom, Atoms
2  from ase.calculators.espresso import Espresso
3  from ase.visualize import view
4  import numpy as np
5  import ase_addons
6
7  bulk = ase_addons.rocksalt(('Fe', 'O'), a=4.3, mags=(0.6, 0))
8  cell10 = bulk.get_cell()
9
10 factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3,
11                    1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0))
12 for factor in factors:
13     atoms = bulk.copy()
14     cell_factor = factor ** (1./3.)
15     atoms.set_cell(cell10 * cell_factor, scale_atoms=True)

```

```

16     with Espresso('FeO-magnetic/FeO-{0:1.2g}V'.format(factor), atoms=atoms,
17                   # CONTROL
18                   calculation='vc-relax',
19                   restart_mode='from_scratch',
20                   disk_io='none',
21                   # SYSTEM
22                   ecutwfc=60.0,
23                   ecutrho=600.0,
24                   occupations='smearing',
25                   smearing='mp',
26                   degauss=0.01,
27                   tot_magnetization = 0,
28                   nspin=2,
29                   # ELECTRONS
30                   mixing_beta=0.3,
31                   conv_thr=1e-8,
32                   # CELL
33                   cell_dofree='shape',
34                   kpts=(6, 6, 6)) as calc:
35         try:
36             calc.calculate()
37         except:
38             raise

```

```

1  from ase import Atom, Atoms
2  from ase.calculators.espresso import Espresso
3  from ase.visualize import view
4  import numpy as np
5  import ase_addons
6
7  bulk = ase_addons.rocksalt(('Fe', 'O'), a=4.3, mags=(0.6, 0))
8  cell0 = bulk.get_cell()
9
10 ready = True
11
12 energies, vols = [], []
13
14 factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3))
15 for factor in factors:
16     atoms = bulk.copy()
17     cell_factor = factor ** (1./3.)
18     atoms.set_cell(cell0 * cell_factor, scale_atoms=True)
19     with Espresso('FeO-magnetic/FeO-{0:1.2g}V'.format(factor), atoms=atoms,
20                   # CONTROL
21                   calculation='vc-relax',
22                   restart_mode='from_scratch',

```



```

23         disk_io='none',
24         # SYSTEM
25         ecutwfc=60.0,
26         ecutrho=600.0,
27         occupations='smearing',
28         smearing='mp',
29         degauss=0.01,
30         tot_magnetization=0,
31         nspin=2,
32         # ELECTRONS
33         mixing_beta=0.3,
34         conv_thr=1e-8,
35         # CELL
36         cell_dofree='shape',
37         kpts=(6, 6, 6)) as calc:
38     try:
39         atoms = calc.get_atoms()
40         vols.append(atoms.get_volume())
41         energies.append(atoms.get_potential_energy())
42     except (AttributeError):
43         print 'AttributeError', calc.espressodir
44         ready = False
45
46 if ready == False:
47     import sys; sys.exit()
48
49 from ase_addons.utils import *
50 eos = EquationOfState(vols, energies)
51 eos.plot('supporting-figures/FeO-eos.png', show=True, title='FeO')

```

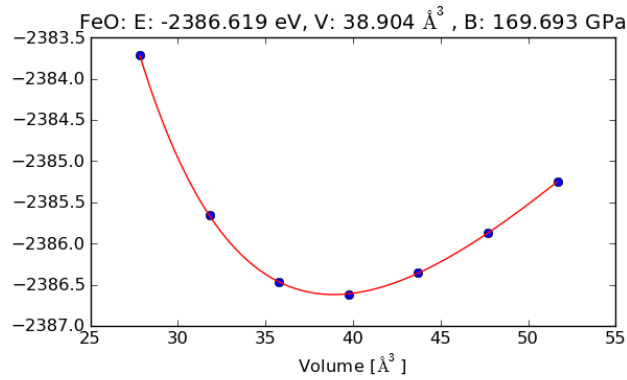


FIG. 29. FeO EOS.

2. Fe_3O_4

```
1 from ase import Atom, Atoms
2 from ase.calculators.espresso import Espresso
3 from ase.visualize import view
4 import numpy as np
5 import ase_addons
6
7 Fe304 = ase_addons.spinel(('Fe', 'Fe', 'O'), a=8.395, mags=(0.6, 0.6, 0))
8 cell0 = Fe304.get_cell()
9
10 factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3,
11                    1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0))
12 for factor in factors:
13     atoms = Fe304.copy()
14     cell_factor = factor ** (1./3.)
15     atoms.set_cell(cell0 * cell_factor, scale_atoms=True)
16     with Espresso('Fe304-magnetic/Fe304-{0:1.2g}V'.format(factor), atoms=atoms,
17                  # CONTROL
18                  calculation='relax',
19                  restart_mode='from_scratch',
20                  disk_io='none',
21                  # SYSTEM
22                  ecutwfc=60.0,
23                  ecutrho=600.0,
24                  occupations='smearing',
25                  smearing='mp',
26                  degauss=0.01,
27                  nspin=2,
28                  # ELECTRONS
29                  mixing_beta=0.3,
30                  conv_thr=1e-8,
31                  kpts=(6, 6, 6)) as calc:
32         try:
33             calc.calculate()
34         except:
35             raise
```

```
1 from ase.calculators.espresso import Espresso
2 import numpy as np
3
4 factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3))
5 energies, vols = [], []
6 for factor in factors:
7     with Espresso('Fe304-magnetic/Fe304-{0:1.2g}V'.format(factor)) as calc:
```

```

8         energies.append(calc.get_potential_energy())
9         vols.append(calc.get_atoms().get_volume())
10
11 from ase_addons.utils import *
12 eos = EquationOfState(vols, energies)
13 eos.plot('supporting-figures/Fe3O4-eos.png', show=True, title='FeO')

```

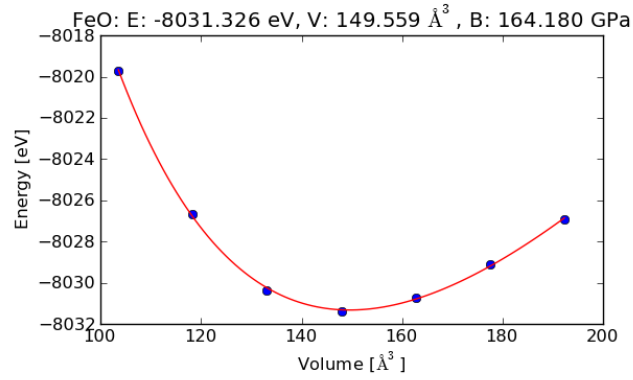


FIG. 30. Fe₃O₄ EOS.

3. Fe₂O₃

```

1 from ase import Atom, Atoms
2 from ase.calculators.espresso import Espresso
3 from ase.visualize import view
4 import numpy as np
5 import ase_addons
6
7 c = 13.8116
8 c_over_a = 2.75
9 z = 0.142
10 x = 0.25 - 0.3125
11
12 chex = c
13 ahex = c/c_over_a
14
15 #Calculating primitive cell
16 a = chex/(3.0*(3.0) ** (0.5)) + ahex/(3.0 * (2.0) ** (0.5))
17 b = ahex*(1.0/(3.0*(2.0) ** 0.5) - 1.0/(2.0) ** 0.5) + chex/(3.0*(3.0) ** 0.5)
18
19 #Primitive lattice vectors

```

```

20  a1 = np.array([b,a,a])
21  a2 = np.array([a,b,a])
22  a3 = np.array([a,a,b])
23
24  bulk = Atoms([Atom('Fe',z*(a1 + a2 + a3), magmom=-0.6),
25               Atom('Fe',-z*(a1 + a2 + a3), magmom=-0.6),
26               Atom('Fe',(0.5 + z)*(a1 + a2 + a3), magmom=0.6),
27               Atom('Fe',(-0.5 - z)*(a1 + a2 + a3), magmom=0.6),
28               Atom('O',x*a1 + (0.5 - x)*a2 + 0.25*a3, magmom=0.00),
29               Atom('O',-x*a1 - (0.5 - x)*a2 - 0.25*a3, magmom=0.00),
30               Atom('O',(0.5 - x)*a1 + 0.25*a2 + x*a3, magmom=0.00),
31               Atom('O',-(0.5 - x)*a1 - 0.25*a2 - x*a3, magmom=0.00),
32               Atom('O',0.25*a1 + x*a2 + (0.5 - x)*a3, magmom=0.00),
33               Atom('O',-0.25*a1 - x*a2 - (0.5 - x)*a3, magmom=0.00)],
34         cell=(a1,a2,a3))
35
36  pos = bulk.get_scaled_positions()
37  bulk.set_scaled_positions(pos %[1,1,1])
38  cell0 = bulk.get_cell()
39
40  factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3,
41                     1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0))
42  for factor in factors:
43      atoms = bulk.copy()
44      cell_factor = factor ** (1./3.)
45      atoms.set_cell(cell0 * cell_factor, scale_atoms=True)
46      with Espresso('Fe203-magnetic/Fe203-{0:1.2g}V'.format(factor), atoms=atoms,
47                   # CONTROL
48                   calculation='vc-relax',
49                   restart_mode='from_scratch',
50                   disk_io='none',
51                   # SYSTEM
52                   ecutwfc=60.0,
53                   ecutrho=600.0,
54                   occupations='smearing',
55                   smearing='mp',
56                   degauss=0.01,
57                   tot_magnetization = 0,
58                   nspin=2,
59                   # ELECTRONS
60                   mixing_beta=0.3,
61                   conv_thr=1e-8,
62                   # CELL
63                   cell_dofree = 'shape',
64                   kpts=(6, 6, 6)) as calc:
65          try:
66              calc.calculate()

```

```

67         except:
68             raise

1  from ase.calculators.espresso import Espresso
2  import numpy as np
3
4  factors = np.array((0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3))
5  energies, vols = [], []
6  for factor in factors:
7      with Espresso('Fe2O3-magnetic/Fe2O3-{0:1.2g}V'.format(factor)) as calc:
8          energies.append(calc.get_potential_energy())
9          vols.append(calc.get_atoms().get_volume())
10
11 from ase_addons.utils import *
12 eos = EquationOfState(vols, energies)
13 eos.plot('supporting-figures/Fe2O3-eos.png', show=True, title='FeO')

```

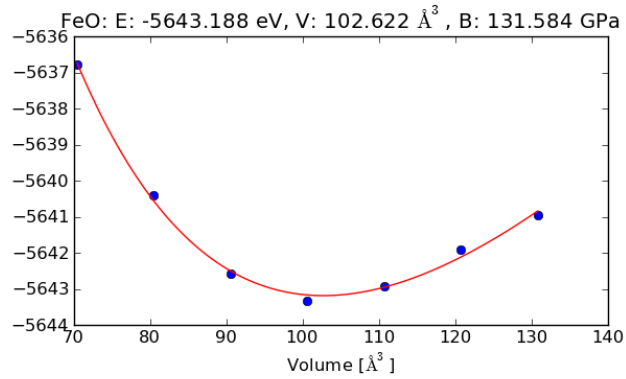


FIG. 31. Fe_2O_3 Equation of State.

D. Nickel oxides

1. NiO

```

1  from espresso import *
2  from ase_addons import rocksalt
3
4  lats = (3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5)
5  energies, vols = [], []
6  ready = True

```

```

7
8  for lat in lats:
9      bulk = rocksalt(('Ni', 'O'), a=lat, mags=(0.7, 0))
10     with Espresso('NiO-QE/EOS/1-{0:1.3f}'.format(lat), atoms=bulk,
11                   calculation='vc-relax',
12                   disk_io='none',
13                   ecutwfc=60.0, ecutrho=600.0,
14                   occupations='smearing', smearing='mp', degauss=0.01,
15                   nspin=2,
16                   conv_thr=1e-8,
17                   cell_dofree='shape',
18                   kpts=(6, 6, 6),
19                   walltime='24:00:00') as calc:
20         try:
21             energies.append(calc.get_potential_energy())
22             vols.append(bulk.get_volume())
23         except (EspressoSubmitted, EspressoRunning):
24             print calc.espressodir, 'running'
25             ready = False
26
27 if ready == False:
28     import sys; sys.exit()
29
30 from ase_addons.utils import *
31 eos = EquationOfState(vols, energies)
32 eos.plot('supporting-figures/NiO-EOS-coarse-QE.png', show=True, title='MnO-coarse')

```

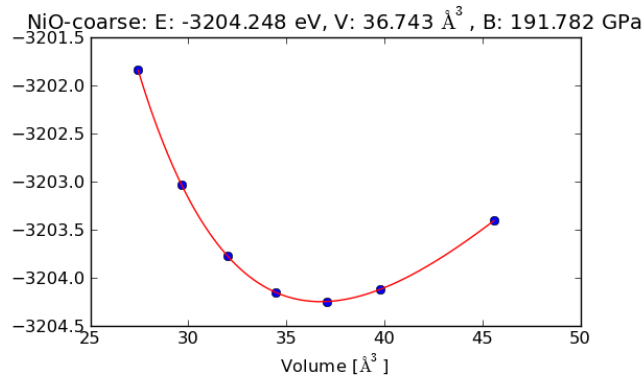


FIG. 32. NiO Equation of State.

E. Cobalt oxides

1. CoO

```
1 from espresso import *
2 from ase_addons import rocksalt
3
4 lats = (3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6)
5 energies, vols = [], []
6 ready = True
7
8 for lat in lats:
9     bulk = rocksalt(('Co', 'O'), a=lat, mags=(0.7, 0))
10    with Espresso('CoO-QE/EOS/1-{:1.3f}'.format(lat), atoms=bulk,
11                  calculation='vc-relax',
12                  disk_io='none',
13                  ecutwfc=60.0, ecutrho=600.0,
14                  occupations='smearing', smearing='mp', degauss=0.01,
15                  nspin=2,
16                  conv_thr=1e-8,
17                  cell_dofree='shape',
18                  kpts=(6, 6, 6),
19                  walltime='24:00:00') as calc:
20        try:
21            energies.append(bulk.get_potential_energy())
22            vols.append(bulk.get_volume())
23        except (EspressoSubmitted, EspressoRunning):
24            print calc.espressodir, 'running'
25            ready = False
26
27 if ready == False:
28     import sys; sys.exit()
29
30 from ase_addons.utils import *
31 eos = EquationOfState(vols, energies)
32 eos.plot('supporting-figures/CoO-EOS-coarse-QE.png', show=True, title='MnO-coarse')
```

2. Co_3O_4

```
1 from ase import Atom, Atoms
2 from ase.calculators.espresso import Espresso
3 from ase.visualize import view
```

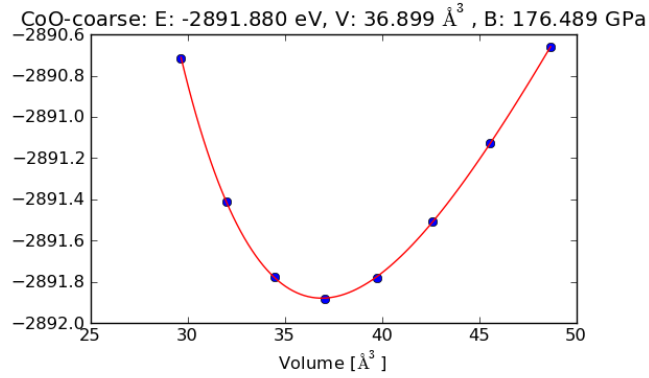


FIG. 33. CoO Equation of State.

```

4  import numpy as np
5  import ase_addons
6
7  Co304 = ase_addons.Co304(('Co', 'Co', 'O'), vol=132, mags=(0.6, 0, 0))
8  cell0 = Co304.get_cell()
9  energies, vols = [], []
10
11  factors = np.array((0.9, 0.95, 1.00, 1.05, 1.10))
12
13  ready = True
14
15  view(Co304)
16  import sys; sys.exit()
17
18  for factor in factors:
19      atoms = Co304.copy()
20      cell_factor = factor ** (1./3.)
21      atoms.set_cell(cell0 * cell_factor, scale_atoms=True)
22      with Espresso('Co304-afm/Co304-{0:1.2f}V'.format(factor), atoms=atoms,
23                  # CONTROL
24                  calculation='relax',
25                  restart_mode='from_scratch',
26                  disk_io='none',
27                  # SYSTEM
28                  ecutwfc=60.0,
29                  ecutrho=600.0,
30                  occupations='smearing',
31                  smearing='mp',
32                  degauss=0.01,
33                  nspin=2,
34                  # ELECTRONS
35                  conv_thr=7e-6,

```



```

36         kpts=(6, 6, 6),
37         ppn=4, mem='8GB') as calc:
38     try:
39         energies.append(calc.get_potential_energy())
40         vols.append(atoms.get_volume())
41     except:
42         print calc.espressodir, 'running'
43         ready = False
44
45 if ready == False:
46     import sys; sys.exit()
47
48 from ase_addons.utils import *
49 eos = EquationOfState(vols, energies)
50 eos.plot('figures/Co3O4-afm-EOS-fine-QE.png', show=True, title='Co3O4-fine')

```

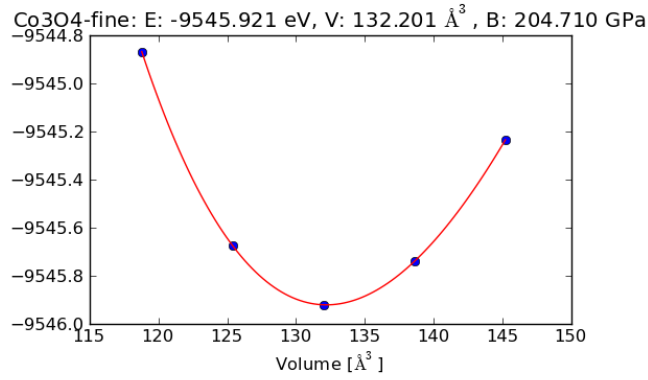


FIG. 34. Co_3O_4 Equation of State.

XI. BEHAVIOR WITH RESPECT TO U FOR ALL SYSTEMS

All calculation parameters used to calculate the $E(U)$ behavior of all materials were the same as those used to determine their corresponding ground state bulk structure, excluding the addition of U . To ensure all bulk structures were in their ground state electronic structures, the approach by Meredig et al. was used [19]. All total energies below are given in eV per metal atom.

A. Original QE PP library

1. *Atoms*

TABLE XXII. Co vs U data.

U (eV)	Total Energy (eV/Co)
0.00	-1007.551
0.05	-1007.545
0.10	-1007.539
0.15	-1007.533
0.20	-1007.528
0.25	-1007.522
0.30	-1007.516
0.35	-1007.511
0.40	-1007.505
0.45	-1007.500
0.50	-1007.494
1.00	-1007.450
1.50	-1007.434
2.00	-1007.434
2.50	-1007.434
3.00	-1007.434
3.50	-1007.434
4.00	-1007.434
4.50	-1007.434
5.00	-1007.434
5.50	-1007.434
6.00	-1007.434
6.50	-1007.434

TABLE XXIII. Cr vs U data.

U (eV)	Total Energy (eV/V)
0.0	-2364.926
0.5	-2364.927
1.0	-2364.928
1.5	-2364.929
2.0	-2364.930
2.5	-2364.931
3.0	-2364.932
3.5	-2364.934
4.0	-2364.935
4.5	-2364.936
5.0	-2364.937
5.5	-2364.938
6.0	-2364.940
6.5	-2364.941
7.0	-2364.942
7.5	-2364.944
8.0	-2364.945
8.5	-2364.946
9.0	-2364.948
9.5	-2364.949
10.0	-2364.950

TABLE XXIV. V vs U data. U (eV) Total Energy (eV/V)

0.0	-369.731
0.1	-369.713
0.2	-369.697
0.3	-369.681
0.4	-369.667
0.5	-369.653
1.0	-369.592
1.5	-369.542
2.0	-369.504
2.5	-369.478
3.0	-369.463
3.5	-369.454
4.0	-369.449
4.5	-369.446
5.0	-369.443
5.5	-369.442
6.0	-369.440
6.5	-369.439

TABLE XXV. Mn vs U data.

U (eV)	Total Energy (eV/Mn)
0.0	-2843.917
0.5	-2843.918
1.0	-2843.920
1.5	-2843.921
2.0	-2843.923
2.5	-2843.925
3.0	-2843.926
3.5	-2843.928
4.0	-2843.930
4.5	-2843.931
5.0	-2843.933
5.5	-2843.935
6.0	-2843.937

TABLE XXVI. Fe vs U data.
 U (eV) Total Energy (eV/Fe)

0.0	-754.538
0.1	-754.517
0.2	-754.496
0.3	-754.477
0.4	-754.460
0.5	-754.443
1.0	-754.370
1.5	-754.304
2.0	-754.255
2.5	-754.234
3.0	-754.217
3.5	-754.202
4.0	-754.186
4.5	-754.171
5.0	-754.156
5.5	-754.142
6.0	-754.128
6.5	-754.114

TABLE XXVII. Ni vs U data.

U (eV)	Total Energy (eV/Ni)
0.0	-1164.040
0.1	-1164.021
0.2	-1164.008
0.3	-1163.998
0.4	-1163.989
0.5	-1163.981
1.0	-1163.947
1.5	-1163.926
2.0	-1163.915
2.5	-1163.908
3.0	-1163.904
3.5	-1163.901
4.0	-1163.899
4.5	-1163.897
5.0	-1163.896
5.5	-1163.894
6.0	-1163.893
6.5	-1163.892
7.0	-1163.891
7.5	-1163.890

2. Metals

TABLE XXVIII. Fe vs U data.

U (eV)	Total Energy (eV/Fe)
0.0	-759.251
0.5	-758.915
1.0	-758.586
1.5	-758.260
2.0	-757.938
2.5	-757.620
3.0	-757.306
3.5	-756.998
4.0	-756.700
4.5	-756.418
5.0	-756.200

TABLE XXIX. V vs U data.

U (eV)	Total Energy (eV/V)
0.0	-376.415
0.5	-375.791
1.0	-375.168
1.5	-374.552
2.0	-373.940
2.5	-373.333
3.0	-372.739
3.5	-372.169
4.0	-371.702
4.5	-371.259
5.0	-370.493

TABLE XXX. Co vs U data.

U (eV)	Total Energy (eV/Co)
0.0	-1012.462
0.5	-1012.168
1.0	-1011.875
1.5	-1011.584
2.0	-1011.294
2.5	-1011.005
3.0	-1010.718
3.5	-1010.437
4.5	-1009.854
5.0	-1009.574

TABLE XXXI. Ni vs U data.

U (eV)	Total Energy (eV/Ni)
0.0	-1168.674
0.5	-1168.511
1.0	-1168.352
1.5	-1168.195
2.0	-1168.041
2.5	-1167.889
3.0	-1167.741
3.5	-1167.595
4.0	-1167.452
4.5	-1167.311
5.0	-1167.173

3. *Oxides*

- Vanadium oxides

TABLE XXXII. VO E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/V)
0.0	-811.845
0.5	-811.470
1.0	-811.116
1.5	-810.782
2.0	-810.458
2.5	-810.146
3.0	-809.843
3.5	-809.550
4.0	-809.264
4.5	-808.986
5.0	-808.713
5.5	-808.447
6.0	-808.187
6.5	-807.932

TABLE XXXIII. V_2O_3 E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/V)
0.0	-1030.248
0.5	-1029.722
1.0	-1029.212
1.5	-1028.722
2.0	-1028.253
2.5	-1027.807
3.0	-1027.381
3.5	-1026.969
4.0	-1026.567
4.5	-1026.172
5.0	-1025.785
5.5	-1025.405
6.0	-1025.030
6.5	-1024.660

TABLE XXXIV. VO₂ E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/V)
0.0	-1247.746
0.5	-1247.170
1.0	-1246.599
1.5	-1246.035
2.0	-1245.479
2.5	-1244.933
3.0	-1244.399
3.5	-1243.779
4.0	-1243.239
4.5	-1242.706
5.0	-1242.179
5.5	-1241.835
6.0	-1241.358
6.5	-1240.887

TABLE XXXV. V_2O_5 E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/V)
0.0	-1464.720
0.5	-1464.108
1.0	-1463.496
1.5	-1462.885
2.0	-1462.275
2.5	-1461.666
3.0	-1461.057
3.5	-1460.451
4.0	-1459.845
4.5	-1459.241
5.0	-1458.638
5.5	-1458.037
6.0	-1457.437
6.5	-1456.840

- Manganese oxides

TABLE XXXVI. MnO E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Mn)
0.0	-3282.409
0.5	-3282.261
1.0	-3282.121
1.5	-3281.986
2.0	-3281.855
2.5	-3281.729
3.0	-3281.608
3.5	-3281.491
4.0	-3281.379
4.5	-3281.270
5.0	-3281.164
5.5	-3281.063
6.0	-3280.964
6.5	-3280.869

TABLE XXXVII. Mn_3O_4 E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Mn)
0.0	-3427.593
0.5	-3427.354
1.0	-3427.123
1.5	-3426.898
2.0	-3426.679
2.5	-3426.467
3.0	-3426.261
3.5	-3426.060
4.0	-3425.865
4.5	-3425.675
5.0	-3425.491
5.5	-3425.312
6.0	-3425.138
6.5	-3424.969

TABLE XXXVIII. MnO₂ E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Mn)
0.0	-3716.733
0.5	-3716.351
1.0	-3715.975
1.5	-3715.604
2.0	-3715.240
2.5	-3714.883
3.0	-3714.535
3.5	-3714.196
4.0	-3713.865
4.5	-3713.543
5.0	-3713.230
5.5	-3712.926
6.0	-3712.632
6.5	-3712.347

- Iron oxides

Note for FeO, we need to change the initial occupancies and provide a slight rhombohedral distortion for Quantum-ESPRESSO to find the ground state structure. A more complete tutorial of how this is done can be found at http://media.quantum-espresso.org/santa_barbara_2009_07/.

TABLE XXXIX. Total energies of rhombohedral FeO with fixed initial occupations.

U (eV)	Total energy (eV/Fe)
0.0	-1193.306
0.5	-1193.007
1.0	-1192.712
1.5	-1192.450
2.0	-1192.231
2.5	-1192.033
3.0	-1191.851
3.5	-1191.679
4.0	-1191.516
4.5	-1191.359
5.0	-1191.208
5.5	-1191.063
6.0	-1190.922
6.5	-1190.785

TABLE XL. Fe_3O_4 E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Fe)
0.0	-1338.562
0.5	-1338.256
1.0	-1337.955
1.5	-1337.657
2.0	-1337.363
2.5	-1337.075
3.0	-1336.790
3.5	-1336.510
4.0	-1336.232
4.5	-1335.958
5.0	-1335.687
5.5	-1335.419
6.0	-1335.155

TABLE XLI. Fe_2O_3 E vs U using Quantum Espresso.

U (eV)	Total energy (eV/Fe)
0.0	-1410.831
0.5	-1410.531
1.0	-1410.238
1.5	-1409.952
2.0	-1409.671
2.5	-1409.397
3.0	-1409.129
3.5	-1408.868
4.0	-1408.613
4.5	-1408.364
5.0	-1408.123
5.5	-1407.888
6.0	-1407.659

- Nickel oxides

TABLE XLII. NiO E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Ni)
0.0	-1602.123
0.5	-1601.992
1.0	-1601.865
1.5	-1601.740
2.0	-1601.616
2.5	-1601.494
3.0	-1601.375
3.5	-1601.257
4.0	-1601.141
4.5	-1601.027
5.0	-1600.914
5.5	-1600.804
6.0	-1600.695
6.5	-1600.588

- Cobalt oxides

TABLE XLIII. CoO E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Co)
0.0	-1445.938
0.5	-1445.665
1.0	-1445.397
1.5	-1445.139
2.0	-1444.933
2.5	-1444.771
3.0	-1444.619
3.5	-1444.472
4.0	-1444.330
4.5	-1444.192
5.0	-1444.058
5.5	-1443.928
6.0	-1443.801
6.5	-1443.678

TABLE XLIV. Co_3O_4 E vs U using Quantum Espresso.

U (eV)	Total Energy (eV/Co)
0.0	-1590.988
0.5	-1590.728
1.0	-1590.476
1.5	-1590.231
2.0	-1589.991
2.5	-1589.755
3.0	-1589.523
3.5	-1589.294
4.0	-1589.068
4.5	-1588.845
5.0	-1588.624
5.5	-1588.405
6.0	-1588.188
6.5	-1587.973

B. GBRV PPs

1. *Atoms*

TABLE XLV. V-atom vs U data.

U (eV)	Total Energy (eV/V)
0.0	-1968.920543
0.5	-1968.789248
1.0	-1968.659247
1.5	-1968.529569
2.0	-1968.400209
2.5	-1968.271136
3.0	-1968.142318
3.5	-1968.013743
4.0	-1967.885398
4.5	-1967.757258
5.5	-1967.501576
6.0	-1967.373995
6.5	-1967.246576
7.0	-1967.119330
7.5	-1966.992240
8.0	-1966.865283
8.5	-1966.738451
9.0	-1966.611742
9.5	-1966.485168
10.0	-1966.358700

TABLE XLVI. Cr-atom vs U data.

U (eV)	Total Energy (eV/Cr)
0.0	-2384.535276
0.5	-2384.485503
1.0	-2384.436840
1.5	-2384.389241
2.0	-2384.342761
2.5	-2384.297339
3.0	-2384.252917
3.5	-2384.209446
4.0	-2384.166823
4.5	-2384.125206
5.0	-2384.084426
5.5	-2384.044541
6.0	-2384.005398
6.5	-2383.967088
7.0	-2383.929599
7.5	-2383.892800
8.0	-2383.856672
8.5	-2383.821328
9.0	-2383.786635
9.5	-2383.752621
10.0	-2383.719273

TABLE XLVII. Mn-atom vs U data.

U (eV)	Total Energy (eV/Mn)
0.0	-2880.048922
0.5	-2880.041731
1.0	-2880.034719
1.5	-2880.027881
2.0	-2880.021198
2.5	-2880.014668
3.0	-2880.008282
3.5	-2880.002036
4.0	-2879.995926
4.5	-2879.989947
5.0	-2879.984094
5.5	-2879.978364
6.0	-2879.972752
6.5	-2879.967254
7.0	-2879.961868
7.5	-2879.956589
8.0	-2879.951415
8.5	-2879.946342
9.0	-2879.941367
9.5	-2879.936487
10.0	-2879.931700

TABLE XLVIII. Fe-atom vs U data.

U (eV)	Total Energy (eV/Fe)
0.0	-3443.653419
0.5	-3443.423007
1.0	-3443.202515
2.0	-3442.788786
2.5	-3442.587911
3.0	-3442.387380
3.5	-3442.187148
4.0	-3441.987219
4.5	-3441.787610
5.0	-3441.588323
5.5	-3441.389368
6.0	-3441.190743
6.5	-3440.992480
7.0	-3440.794569
7.5	-3440.597039
8.0	-3440.399905
8.5	-3440.203181
9.0	-3440.006893
9.5	-3439.811063
10.0	-3439.615721

TABLE XLIX. Co-atom vs U data.

U (eV)	Total Energy (eV/Co)
0.0	-4056.041565
0.5	-4055.967525
1.0	-4055.908426
1.5	-4055.865500
2.0	-4055.838637
2.5	-4055.691739
3.0	-4055.634161
4.5	-4055.441142
5.0	-4055.375820
5.5	-4055.310453
6.0	-4055.245119
6.5	-4055.179837
7.0	-4055.114626
7.5	-4055.049488
8.0	-4054.984438
8.5	-4054.919455
9.0	-4054.854568
9.5	-4054.789769
10.0	-4054.725057

TABLE L. Ni-atom vs U data.

U (eV)	Total Energy (eV/Ni)
0.0	-4675.286298
0.5	-4675.148207
1.0	-4675.015678
1.5	-4674.890559
2.0	-4674.776033
2.5	-4674.677311
3.0	-4674.603363
3.5	-4674.566613
4.0	-4674.561056
4.5	-4674.562355
5.0	-4674.563938
5.5	-4674.565606
6.0	-4674.567354
6.5	-4674.569178
7.0	-4674.571078
7.5	-4674.573048
8.0	-4674.575089
8.5	-4674.577197
9.0	-4674.579370
9.5	-4674.581606
10.0	-4674.583903

2. *Metals*

TABLE LI. V vs U data.

U (eV)	Total Energy (eV/V)
0.00	-1974.11630
0.50	-1973.49349
1.00	-1972.87111
1.50	-1972.24928
2.00	-1971.62817
2.50	-1971.00801
3.00	-1970.38896
3.50	-1969.77106
3.59	-1969.65995

TABLE LII. Mn vs U data.

U (eV)	Total Energy (eV/Mn)
0.00	-2883.57998
0.50	-2883.01708
1.00	-2882.46160
1.50	-2881.90899
2.00	-2881.36060
2.50	-2880.81656
3.00	-2880.27733
4.00	-2880.04058
4.50	-2879.79123
5.00	-2879.55950
5.50	-2879.34278
5.94	-2879.16218

TABLE LIII. Fe vs U data.
 U (eV) Total Energy (eV/Fe)

0.00	-3448.13726
0.50	-3447.60491
1.00	-3447.07453
1.50	-3446.54645
2.00	-3446.02152
2.50	-3445.50239
3.00	-3444.99524
3.50	-3444.50011
4.00	-3444.01900
4.50	-3443.63774
5.00	-3443.27554
5.22	-3443.12086

TABLE LIV. Co vs U data.
 U (eV) Total Energy (eV/Co)

0.00	-4061.16399
0.50	-4060.82795
1.00	-4060.49625
1.50	-4060.16868
2.00	-4059.84387
2.50	-4059.52113
3.00	-4059.20032
3.50	-4058.88301
4.00	-4058.57311

TABLE LV. Ni vs U data.
 U (eV) Total Energy (eV/Ni)

0.00	-4679.96202
0.50	-4679.71085
1.00	-4679.46255
1.50	-4679.21712
2.00	-4678.97450
2.50	-4678.73464
3.00	-4678.49780
3.50	-4678.26458
4.00	-4678.03497
4.50	-4677.80835
5.00	-4677.58432
5.50	-4677.36287
6.00	-4677.14395
6.50	-4676.92751
7.00	-4676.71365
7.50	-4676.50271
7.69	-4676.42338

TABLE LVI. Cr vs U data.
 U (eV) Total Energy (eV/Cr)

0.00	-2388.46391
0.50	-2387.85385
1.00	-2387.24593
1.50	-2386.64063
2.00	-2386.03854
2.50	-2385.44043
3.00	-2384.84743
3.50	-2384.26049
4.00	-2383.68051
4.50	-2383.10904
5.00	-2382.54819
5.50	-2382.00823
6.00	-2381.49120
6.02	-2381.47091

3. Oxides

- Vanadium oxides

TABLE LVII. VO vs U data.
 U (eV) Total Energy (eV/V)

0.00	-2414.88969
0.50	-2414.51033
1.00	-2414.15053
1.50	-2413.80936
2.00	-2413.47922
2.50	-2413.16064
3.00	-2412.85203
3.50	-2412.55219
3.61	-2412.48731

TABLE LVIII. V_2O_3 vs U data.

U (eV)	Total Energy (eV/V)
0.00	-2635.91572
0.50	-2635.39206
1.00	-2634.88598
1.50	-2634.39963
2.00	-2633.93593
2.50	-2633.49525
3.00	-2633.07448
3.50	-2632.66701
4.00	-2632.34813
4.50	-2631.96071
4.64	-2631.85346

TABLE LIX. VO_2 vs U data.

U (eV)	Total Energy (eV/V)
0.00	-2856.14360
0.50	-2855.57010
1.00	-2855.00108
1.50	-2854.43763
2.00	-2853.88086
2.50	-2853.33419
3.00	-2852.80734
3.50	-2852.29306
4.00	-2851.78854
4.50	-2851.29862
5.00	-2850.81790
5.02	-2850.79885

TABLE LX. V_2O_5 vs U data.

U (eV)	Total Energy (eV/V)
0.00	-3075.80618
0.50	-3075.19515
1.00	-3074.58488
1.50	-3073.97544
2.00	-3073.36690
2.50	-3072.75934
3.00	-3072.15285
3.50	-3071.54752
4.00	-3070.94346
4.50	-3070.34240
4.67	-3070.13782

- Chromium oxides

TABLE LXI. Cr_2O_3 vs U data.

U (eV)	Total Energy (eV/Cr)
0.00	-3049.51732
0.50	-3049.19037
1.00	-3048.87190
1.50	-3048.56100
2.00	-3048.25635
2.50	-3047.95721
3.00	-3047.66392
3.50	-3047.37428
4.00	-3047.08890
4.50	-3046.80751
4.86	-3046.60728

- Manganese oxides

TABLE LXII. MnO vs U data.
 U (eV) Total Energy (eV/Mn)

0.00	-3323.89011
0.50	-3323.75702
1.00	-3323.63160
1.50	-3323.51284
2.00	-3323.39998
2.50	-3323.29242
3.00	-3323.19012
3.50	-3323.09171
4.00	-3322.99748
4.50	-3322.90715
5.00	-3322.82049
5.50	-3322.73729
5.52	-3322.73403

TABLE LXIII. Mn_3O_4 vs U data.

U (eV)	Total Energy (eV/Mn)
0.00	-3470.89980
0.50	-3470.67106
1.00	-3470.45120
1.50	-3470.23928
2.00	-3470.03440
2.50	-3469.83604
3.00	-3469.64374
3.50	-3469.45714
4.00	-3469.27600
4.50	-3469.10014
5.00	-3468.92938
5.50	-3468.76368
6.00	-3468.60268
6.11	-3468.56790

TABLE LXIV. MnO_2 vs U data.

U (eV)	Total Energy (eV/Mn)
0.00	-3763.64640
0.50	-3763.27745
1.00	-3762.91426
1.50	-3762.55701
2.00	-3762.20604
2.50	-3761.86176
3.00	-3761.52476
3.50	-3761.19586
4.00	-3760.87586
4.20	-3760.75046

- Iron oxides

TABLE LXV. FeO vs U data.

U (eV)	Total Energy (eV/Fe)
0.00	-3888.01905
0.50	-3887.74096
1.00	-3887.46801
1.50	-3887.20191
2.00	-3886.98152
2.50	-3886.80535
3.00	-3886.65125
3.50	-3886.51155
4.00	-3886.38277
4.50	-3886.26303
5.00	-3886.15003
5.50	-3886.04314
5.80	-3885.98163

TABLE LXVI. Fe_3O_4 vs U data.

U (eV)	Total Energy (eV/Fe)
0.00	-4035.10385
0.50	-4034.83523
1.00	-4034.57446
1.50	-4034.32047
2.00	-4034.07278
2.50	-4033.83189
3.00	-4033.59767
3.50	-4033.36909
4.00	-4033.14530
4.50	-4032.92593
5.00	-4032.71076
5.50	-4032.49966
6.00	-4032.29250
6.07	-4032.26380

TABLE LXVII. Fe_2O_3 vs U data.

U (eV)	Total Energy (eV/Fe)
0.00	-4108.28747
0.50	-4108.02180
1.00	-4107.76946
1.50	-4107.52820
2.00	-4107.29537
2.50	-4107.07190
3.00	-4106.85588
3.50	-4106.64663
4.00	-4106.44509
4.50	-4106.24780
5.00	-4106.05587
5.21	-4105.97678

- Cobalt oxides

TABLE LXVIII. CoO vs U data.

U (eV)	Total Energy (eV/Co)
0.00	-4499.87220
0.50	-4499.60017
1.00	-4499.33954
1.50	-4499.14282
2.00	-4498.99682
2.50	-4498.86127
3.00	-4498.73322
3.50	-4498.61161
4.00	-4498.49558
4.50	-4498.38454
5.00	-4498.27809
5.50	-4498.17594
5.86	-4498.10492

TABLE LXIX. Co_3O_4 vs U data.

U (eV)	Total Energy (eV/Co)
0.00	-4646.86221
0.50	-4646.60676
1.00	-4646.36220
1.50	-4646.12488
2.00	-4645.89406
2.50	-4645.66862
3.00	-4645.44774
3.50	-4645.23087
4.00	-4645.01737
4.50	-4644.80599
5.00	-4644.59821
5.50	-4644.39293
6.00	-4644.18997
6.50	-4643.98920
7.00	-4643.79052
7.44	-4643.61734

- Nickel oxides

TABLE LXX. NiO vs U data.
 U (eV) Total Energy (eV/Ni)

0.00	-5118.69295
0.50	-5118.55616
1.00	-5118.42757
1.50	-5118.30440
2.00	-5118.18576
2.50	-5118.07178
3.00	-5117.96228
3.50	-5117.85665
4.00	-5117.75488
4.50	-5117.65680
5.00	-5117.56233
5.50	-5117.47139
6.00	-5117.38392
6.50	-5117.29989
7.00	-5117.21924
7.50	-5117.14196
7.59	-5117.12840

XII. REQUIRED MODULES

We used the Atomic Simulation Environment (ASE), which can be downloaded at <https://wiki.fysik.dtu.dk/ase/>. In addition, the standard python scientific computing modules of `numpy`, `scipy`, and `matplotlib` were used. The modules below were custom made by John R. Kitchen and Zhongnan Xu and are required to generate the code and results in this paper.

A. espresso

`espresso` is the quantum-espresso wrapper Zhongnan Xu wrote to integrate ASE with the QUANTUM-ESPRESSO package. A most recent version of the code can be found at <https://github.com/zhongnanxu/espresso>.

B. ase_addons

The `ase_addons` module is a ASE addons module containing code for constructing surfaces, bulk crystal structures, etc. It can be found at https://github.com/zhongnanxu/ase_addons.

C. pycse.py

Python computations in science and engineering (PYCSE) contains the statistical tools needed for the data analysis portion. This can be found for free at <https://github.com/jkitchin/pycse>.

* jkitchin@andrew.cmu.edu

- [1] We used the pseudopotentials O.pbe-rrkjus.UPF, V.pbe-n-van.UPF, Cr.pbe-sp-van.UPF, Mn.pbe-sp-van.UPF, Fe.pbe-nd-rrkjus.UPF, Co.pbe-nd-rrkjus.UPF, and Ni.pbe-nd-rrkjus.UPF from <http://www.quantum-espresso.org>.
- [2] K. F. Garrity, J. W. Bennett, K. M. Rabe, and D. Vanderbilt, *Comp. Mater. Sci.* **81**, 446 (2014).
- [3] A. Rata and H. T, *Strain-Induced Properties of Epitaxial VO_x Thin Films*, Ph.D. thesis, Rijksuniversiteit Groningen (2004).
- [4] E. Z. Kurmaev, V. Cherkashenko, Y. M. Yarmoshenko, S. Bartkowski, A. Postnikov, M. Neumann, L. C. Duda, J. Guo, J. Nordgren, V. Perelyaev, *et al.*, *J. Phys.-Condensed Matter* **10**, 4081 (1998).
- [5] A. E. Bocquet, T. Mizokawa, K. Morikawa, A. Fujimori, S. R. Barman, K. Maiti, D. D. Sarma, Y. Tokura, and M. Onoda, *Phys. Rev. B* **53**, 1161 (1996).

- [6] W. Strehlow and E. Cook, [J. Phys. Chem. Ref. Data](#) **2**, 163 (1973).
- [7] Z. Hanafi, F. Ismail, and A. Mohamed, [Z. Phys. Chem.](#) **194**, 61 (1996).
- [8] A. Jha, R. Thapa, and K. Chattopadhyay, [Mater. Res. Bull.](#) **47**, 813 (2012).
- [9] H. Bowen, D. Adler, and B. Auker, [J. Solid. State. Chem.](#) **12**, 355 (1975).
- [10] A. Chainani, T. Yokoya, T. Morimoto, T. Takahashi, and S. Todo, [Phys. Rev. B](#) **51**, 17976 (1995).
- [11] R. Zimmermann, P. Steiner, R. Claessen, F. Reinert, S. Hüfner, P. Blaha, and P. Dufek, [J. Phys.-Condensed Matter](#) **11**, 1657 (1999).
- [12] J. van Elp, J. L. Wieland, H. Eskes, P. Kuiper, G. A. Sawatzky, F. M. F. de Groot, and T. S. Turner, [Phys. Rev. B](#) **44**, 6090 (1991).
- [13] O. Kubaschewski, C. Alcock, and P. Spencer, *Materials thermochemistry*, Vol. 6 (Pergamon press Oxford, 1993).
- [14] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. a. Persson, [APL Materials](#) **1**, 011002 (2013).
- [15] G. Hautier, S. P. Ong, A. Jain, C. J. Moore, and G. Ceder, [Phys. Rev. B](#) **85**, 155208 (2012).
- [16] V. L. Chevrier, S. P. Ong, R. Armiento, M. K. Y. Chan, and G. Ceder, [Phys. Rev. B](#) **82**, 075122 (2010).
- [17] C. Kittel, *Introduction to Solid State Physics*, 6th ed. (John Wiley & Sons, Inc., New York, 1986).
- [18] A. B. Alchagirov, J. P. Perdew, J. C. Boettger, R. C. Albers, and C. Fiolhais, [Phys. Rev. B](#) **67**, 026103 (2003).
- [19] B. Meredig, A. Thompson, H. A. Hansen, C. Wolverton, and A. van de Walle, [Phys. Rev. B](#) **82**, 195128 (2010).